Computer
Firmware
Security
Technology

计算机固件安全技术

周振柳 著
Zhou Zhenliu

清华大学出版社

计算机固件安全技术

Computer Firmware Security Technology

周振柳　著

Zhou Zhenliu

清华大学出版社

北　京

内 容 简 介

本书内容涵盖作者在计算机固件安全领域多年的研究成果，是国内第一部公开出版的计算机固件安全领域的学术著作。

全书内容包括：计算机固件概念和功能、国内外固件产品和技术研究发展历程、固件开发基础技术与规范、固件安全研究历史与现状、传统固件 BIOS 安全技术研究开发实例、BIOS 安全漏洞及其威胁、BIOS 安全检测方法与实现、可信固件开发的安全策略和模型、可信固件中可信度量基础与方法、可信固件的开发实现。

本书可作为高等学校网络安全、信息安全专业教材，或相关专业人员的参考研究书籍。

图书在版编目（CIP）数据

计算机固件安全技术/周振柳著.--北京：清华大学出版社，2012.12
ISBN 978-7-302-30111-0

Ⅰ.①计…　Ⅱ.①周…　Ⅲ.①固件—安全技术　Ⅳ.①TP303

中国版本图书馆 CIP 数据核字(2012)第 217798 号

责任编辑：梁　颖
封面设计：
责任校对：李建庄
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×230mm　　**印　　张：**12.25　　**字　　数：**269 千字
版　　次：2012 年 12 月第 1 版　　**印　　次：**2012 年 12 月第1 次印刷
印　　数：1～　000
定　　价：　.00 元

产品编号：049790-01

前　言

FOREWORD

计算机固件 BIOS 的发展历史并不长，它诞生于 1981 年出产的第一台个人计算机中。尽管固件 BIOS 产品一直是计算机中最底层、最核心的关键性部件之一，但却由于其位处后台，默默无闻，而少人关注。而在国内，BIOS 产品则成为我国计算机产业链中长期缺失的一环，只能依赖我国台湾地区和国外的产品。

1995 年初始，笔者有幸参与"FTPC-8086 柔性实验教学系统"的开发，负责下位机系统软件的编写工作，主要就是从开机上电的第一条指令开始编写，完成各硬件部件测试例程、单步中断(int 1)例程、断点中断(int 3)例程，以及其他常用的 BIOS 中断和 DOS 中断例程，并将例程执行结果通过串口通信返回给上位机上的集成调试运行环境，以使这些例程能被上位机运行的程序调用执行。其中下位机上从开机上电的第一条指令开始，每一个例程，都需要自己独立编写完成，工作的实质就是要写一个 Mini BIOS。因此不得不查阅学习了许多 BIOS 底层和 DOS 底层相关技术，并精熟了汇编语言程序设计。自此体会到底层固件开发的挑战和乐趣，也经常愿意在专业课教学中向学生炫耀这一段开发历程。这项工作，为笔者后来步入计算机固件安全研究领域奠定了技术基础。

大概在 2002 年 3 月的时候，刘军找到我，跟我讲了美国 BIOS 厂商 Phoenix 公司的 Phoenix. net 产品的事情，希望我能够对它研究一下。Phoenix. net 是 Phoenix 公司的一个 BIOS 产品项目，计划在计算机底层 BIOS 固件中嵌入一个"音序器"，指导并帮助计算机用户完成在操作系统安装后烦琐的应用软件下载和安装工作。笔者因此采用了逆向工程的方法，对 Phoenix. net 固件产品进行了研究剖析并提交了技术分

析报告。这项工作，向笔者揭开了计算机固件层次安全的神秘面纱，自此发现计算机固件安全的别样洞天，成为笔者步入计算机固件安全研究领域的契机。后来一系列BIOS安全增强、BIOS安全代理、BIOS木马研究工作，都自此展开。非常怀念这一段美好神奇的研究开发之旅。

2005年，笔者有幸拜入到我国著名网络安全专家许榕生先生门下，成为先生的博士学生。鉴于国内对计算机固件安全研究关注甚少，先生建议我对计算机固件安全开展系统性的研究工作。于是深入挖掘固件安全漏洞，研究其安全检测原理和方法。时值信息产业部(现工业与信息化部)在幕后者多年的努力推动下，终于启动并支持“新一代安全BIOS的研制和产业化”、“高安全与可管理BIOS”等研究项目工作，笔者也参与其中，开展安全BIOS和可信固件的研究开发工作。

在以上历程中，每每感叹国内计算机固件BIOS领域资料的缺乏。时至今日，也只有台湾地区旗标出版社出版的陈文钦著《BIOS Inside BIOS研发技术剖析》一书，可算国内唯一能找到的中文BIOS技术宝典，而涉及计算机固件安全的公开出版物，则尚未得见。有感如此，深感补此遗缺，乃我辈之责任。遂成书，以飨读者。

本书共9章，其内容可分为三部分。第一部分讲述计算机固件技术发展历程、安全研究现状、技术基础，包括第1章和第2章；第二部分阐述传统固件BIOS的安全技术、安全漏洞及其安全检测方法和系统，包括第3～5章；第三部分从可信理论出发，研究基于新一代EFI固件的可信固件安全策略模型、可信度量基础和方法以及可信固件的开发实现，包括第6～8章；最后是对本书研究内容的总结和展望。

本书成书及出版，得到众多热心人的支持和帮助。在本书最后部分的致谢中，笔者要对他们致以衷心的感激之情。特别地要感谢清华出版社的编辑梁颖老师对本书出版的支持和跟踪追进。

周振柳

2012年7月于沈阳

目 录 CONTENTS

CONTENTS 目 录

目 录 CONTENTS

CONTENTS 目 录

目 录 CONTENTS

Computer Firmware Security Technology

第 1 章
引 言

计算机传统固件 BIOS 产品，由于其深处计算机运行后台，用户可见的运行周期少至只有几十秒，因此不为计算机用户所熟知。而传统固件 BIOS 技术，IBM 设计之初是开放的状态，但由于其底层地位和纽带作用，后来的发展陷入到封闭状态，只为少数几个厂商所掌握，外人知之甚少。基于这些原因，计算机固件安全这个领域，在国内很长一段时期内被忽视，无人瞩目。

本章介绍计算机固件产品的概念、功能和作用，简单叙述从传统 BIOS 到 EFI 固件的发展历程，从中引出固件 BIOS 安全研究的背景、历史与现状。以期引起对计算机固件安全研究的兴趣和重视。

1.1 固件在计算机中的地位和作用

计算机固件是计算机系统中不可缺少的底层基础系统。这些固件往往是以软件的形式固化存储在硬件芯片中。计算机主板上有最重要、最核心的计算机固件，通常称为 BIOS(Basic Input/Output System，基本输入输出系统)，计算机加电时，中央处

理器(Central Processing Unit,CPU)取得并执行的第一条指令,就存储在 BIOS 固件中。因此,BIOS 固件首先取得计算机运行控制权,其后的硬件系统初始化和检测都由 BIOS 固件完成,最后才加载计算机操作系统,并把系统控制权移交给操作系统。但由于 BIOS 固件处在软件的最底层,很少直接同用户交互,因此往往被大多数人忽视。

1.1.1 固件和 BIOS 的概念

固件(Firmware)是一种底层软件或指令序列,早期常常存储于计算机或外部设备的可编程只读存储器(Programable Read-Only Memory,PROM)中。固件常用于对硬件设备进行配置,使得硬件设备能够完成指定的功能,或者为操作系统提供硬件操作接口。一些固件也可能成为操作系统内核的一部分,为操作系统的其他部分提供基本服务,并运行于特权模式下,这种情况多见于嵌入式系统和设备中。在计算机系统中,BIOS 就是一种核心固件。计算机系统中其他的一些外围设备(如显示卡、网卡等)上也存在着固件,用于初始化、驱动这些设备,为操作系统提供操作接口。

1981 年,IBM 在设计第一部个人计算机——IBM PC 时,工程师把一些开机时的硬件初始化/检测代码,从软盘或硬盘装载操作系统、完成开机程序的前导代码,以及一些最基本的外围设备处理的代码(如屏幕显示、磁盘驱动等),压缩存入一颗大约 32KB 大小的 PROM[1],这些代码组成了 BIOS。

当计算机开机的一瞬间,硬件特性就是设计成 CPU 从主板上 BIOS 芯片内取得指令码(对于 16 位计算机这个地址是 0FFFF0,对于 32 位计算机这个地址是 0FFFFFF0),BIOS 内部的指令取得系统控制权,然后再跳转到 BIOS 固件其他位置的指令去执行。经过 CPU 的检测设定、配置内存、初始化南北桥芯片组,最后驱动磁盘把操作系统载入内存,将系统控制权转移给操作系统引导代码(OS Loader),BIOS 的开机引导工作就告一段落,改为从事幕后对操作系统的支持、协调工作,帮助操作系统或应用程序来处理外围设备的执行动作。

事实上除了主板上的 BIOS 外,另外如显卡、SCSI 卡、网卡、硬盘控制器等也都有

各自的BIOS芯片。设计在主板上、负责整个主板运行的BIOS，一般称为系统BIOS(System BIOS)或主板BIOS(Mainboard BIOS)，如果没有特别声明，BIOS所指就是主板上的BIOS；而显卡上的BIOS一般称为Video BIOS，SCSI控制卡上的BIOS则以SCSI BIOS称呼。这种扩展卡或控制器上的BIOS后来也统称为OPROM(Option ROM)。

这种固件早期都存储在PROM芯片中，只能用特定的编程器对其进行读写操作，不能在主板等设备中直接对其进行读写操作；其容量也逐渐从8KB、16KB、32KB、64KB增加到128KB、256KB和512KB不等。从大约1996年以后，伴随flash芯片技术的发展，以及对BIOS固件在线更新的要求，以及计算机系统使用后期要求能够从BIOS固件更新CPU微代码(Micro Code)，从而实现为CPU打补丁功能的出现，计算机系统中的固件存储逐渐从ROM芯片转换成flash芯片。其特征是：①芯片容量继续增大，以适应越来越多的应用加入到固件中的要求，②允许在操作系统运行过程中，通过指定技术直接对固件芯片中的内容进行更新写入。在为用户和系统维护带来更多的方便功能的同时，也就此引入了更多的安全风险。

1.1.2 固件功能及地位

计算机系统层次架构由硬件(Hardware)、固件(Firmware)、软件(Software)构成，传统上计算机固件的核心就是BIOS，计算机软件的核心是操作系统，如图1.1所示。在这种层次架构中，BIOS介于硬件和操作系统之间，为操作系统提供最直接、最底层的硬件控制，为操作系统的引导载入和正常运行提供良好支持。这种架构的好

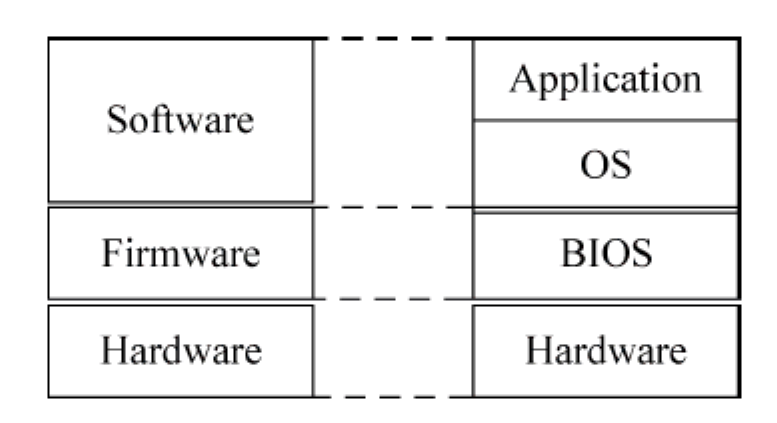


图1.1 计算机系统层次架构

处之一，是 BIOS 能够为操作系统提供一定程度的硬件抽象层，使操作系统可以不涉及具体的硬件操作，具有更好的平台无关性。

BIOS 是计算机的核心部件之一，计算机开机以后执行的第一条指令就由 BIOS 发出。BIOS 担负着系统最初的引导和启动任务，为操作系统对计算机系统硬件设备进行管理提供基础服务。没有 BIOS，计算机系统的所有硬件就无法工作。BIOS 的损坏直接导致计算机系统可用性遭到破坏，只能通过芯片级别的硬件维修才能够恢复。

BIOS 的主要功能可以概括为下面几点[1,2,3]：

（1）**开机自检（Power On Self Test，POST）** 计算机开机时系统将控制权交给 BIOS，BIOS 针对 CPU 各项寄存器、标志位等先检查 CPU 是否工作正常，接下来会检查 8254 定时器、8259A 可编程中断控制器、8237DMA 控制器等的状态，测试其是否工作正常。这些自检还包括 640KB 的基本内存、串并口、键盘、显卡等。一旦在自检过程中发现问题，BIOS 将给出提示信息或鸣笛警告。

（2）**系统初始化** 针对 CPU Cache、DRAM、南北桥芯片组、显卡、PCI 设备控制器、IDE 设备控制器、网卡等的寄存器作初始化操作，填充相应寄存器的值，设定成可支持的默认工作模式，并检测是否能够正常工作。查找并运行外围设备上存在的 Option ROM 使其对外围设备进行驱动。

（3）**提供常驻内存的运行时服务（Runtime Services）** 这些服务程序常驻在某一段系统内存中，操作系统和应用程序能够通过中断方式调用这些服务代码，典型的如 Int 10h、Int 13h、Int 15h 等。这些常驻内存的运行时服务使得系统控制权由 BIOS 转移给操作系统后，BIOS 代码仍然可以在适当的时机被执行。

（4）**系统设置** BIOS 提供文本或图形界面的设置程序（通常称为 BIOS Setup），供用户在进入操作系统前对系统的一些参数值进行设置，如 BIOS 密码、光盘/硬盘/软盘引导顺序、系统时间等。这些参数设置记载在非易失性存储体（Non-Volatile RAM）中，如 CMOS（Complementary Metal Oxide Semiconductor，互补金属氧化物半导体）芯片或 flash 芯片的扩展系统配置数据区（Extended System Configuration Data，ESCD）。

(5) **引导操作系统** BIOS在执行的最后阶段,会按照设置中保存的启动顺序搜索软硬盘驱动器及CD-ROM、网络服务器等,查找到有效的启动记录后,读入操作系统引导代码,然后将系统控制权交给引导代码,并由引导代码来完成操作系统的装载启动。

1.2 相关领域国内外研究现状和趋势

在计算机系统中,固件处于一个很不显眼的位置,并且较少同用户直接交互。但固件却是计算机系统中不可或缺的底层软件系统,如果固件系统的安全得不到保障,则其上层操作系统的安全以及应用软件层的安全保障措施都可能失效。伴随固件和安全技术的发展,固件在信息安全中的重要性逐渐呈现出来[4]。

1.2.1 计算机固件产品研发现状和趋势

从1981年第一个BIOS产品出现以来,BIOS技术和产品的发展同IT行业其他领域发展相比较,呈现一个明显的滞后状态。在计算机中央处理器、外围部件和操作系统从8位到16位、32位、64位的快速发展过程中,BIOS技术则从最初的8位发展到16位后一直停滞不前。这里有两个重要的原因:一是BIOS技术处于底层,只需要完成对平台硬件初始化,引导操作系统即可,很少直接同用户交互,因此推动其发展的动力不足;二是BIOS技术和市场的垄断,使得BIOS厂商为保护自己的既得利益,不愿意去推动BIOS新技术和新产品的发展。

1998年,Intel公司开始封闭开发一种专用于其服务器的64位BIOS产品。2003年,Intel在将该新一代BIOS产品成功地应用于Intel 64位服务器的基础上,发布了32位和64位新一代BIOS产品的新规范——扩展固件接口(Extensible Firmware Interface,EFI)[5],并联合业界Microsoft、IBM、Hp、AMI、Dell、Phoenix、AMD等厂商推动该规范成为下一代BIOS的行业规范——Unified EFI(UEFI)[6,7]。通过随后几

年的推动，目前 UEFI 已经成为新一代 BIOS 的行业标准，并且到 2007 年，Intel 已经将其生产的所有主板、PC 和服务器上的 BIOS 换成符合 UEFI 标准的新一代 BIOS 产品。BIOS 厂商 Insyde 和 AMI 也已经推出新的符合 UEFI 规范的 BIOS 产品。

从 BIOS 产品市场来看，公用 BIOS 产品一直由我国台湾地区和美国公司所垄断。1998 年之前，全球公用 BIOS 市场由美国 Phoenix、AMI 和我国台湾地区 Award 三家垄断，1998 年之后则是美国 Phoenix、AMI 和我国台湾地区 Insyde 三家垄断（Award 公司被 Phoenix 公司所收购，以弥补其在 PC 固件产品中市场占有不足的局面）。欧洲企业共同制定的 Open Firmware 规范由于缺乏 BIOS 厂商和芯片组厂商的技术支持，也只能在较小范围的专用服务器上形成可用产品。而一些开源 BIOS 项目如 LinuxBIOS[8,9]、TinyBIOS 也由于同样的原因只能用于少数定制的硬件平台和操作系统。由于牵扯到专利权与厂商之间的竞争因素，BIOS 产品的兼容性与合法性一直是试图进入这一领域的企业要面对的严肃问题。另外，芯片组技术和资料的垄断，也是形成 BIOS 技术和产品垄断的一个重要因素。

我国信息技术产业在 BIOS 领域长期以来处于空白局面，国内 PC 和服务器 BIOS 完全依赖上述三家 BIOS 厂商提供产品和服务。这不仅影响我国建立完整的信息产业链，对国家的信息安全来讲也是一个不可忽视的隐患。借助新一代 BIOS 发展的契机，2005 年我国信息产业部通过电子产业发展基金支持，启动“新一代安全 BIOS 的研制和产业化”、“高安全与可管理 BIOS”等项目，推动自主研发适合我国国情的 BIOS 产品的研究工作。其中获得 Intel 公司技术合作支持的包括中国电子科技集团信息化工程总体研究中心和南京百敖软件有限公司（BYOSOFT）。

1.2.2 BIOS 安全研究历史与现状

国内外对安全操作系统 30 多年的持续研究，在理论方法、技术和产品上都取得了丰富的成果[10,11,12,13,14,15,16]，而对计算机 BIOS 系统的安全研究 20 多年来则几乎被忽略。William A. Arbaugh 等在 1997 年提出一种计算机安全引导架构 AEGIS[17]。AEGIS 基于 IBM PC 传统 BIOS，采用认证的方法保障 BIOS 固件代码的完整性，增

强 BIOS 引导过程中 BIOS 代码的安全保护。但 AEGIS 缺乏硬件保护措施，也没有考虑固件层对系统软件层的延伸保护。1998 年，Dexter Kozen 在基于语言安全研究[18,19,20,21,22]的基础上提出一种简化高效的语言证明方法 ECC[23]，使用编译技术在代码执行前检查代码控制流、内存访问以及堆栈操作的安全逻辑，并针对 Open Firmware 语言和指令集进行了优化和实现。2002 年，Frank Adelstein、Matt Stillerman 和 Dexter Kozen 演示了在 Open Firmware 中如何利用 ECC 方法进行固件的恶意代码检测[24]。2003 年，Matt Stillerman 和 Dexter Kozen 开发了基于 ECC 的安全 Open Firmware 的原型系统[25]，由于语言证明方法的复杂性，这种方法并不成熟，离实际应用尚有较大差距。

而从 1998 年开始，针对 BIOS 系统的公开性的安全事件和威胁时有发生。1998 年 7 月开始爆发的 CIH 病毒就实施了对计算机 BIOS 的攻击，造成全球大量 PC 不能使用。1999 年和 2000 年，美国 BIOS 厂商 Phoenix 公司的 PhoenixNet 产品在 BIOS 中嵌入的 ILS 模块具有远程安装、控制、回传、定位等功能，涉嫌控制用户计算机和暴露用户隐私，遭到广泛的抵制和批评[4]。2006 年全球 Black Hat 会议上，英国 Next-Generation 安全软件公司的首席安全顾问 John Heasman 阐述了一种新的 rootkit 技术[26]，在计算机主板 BIOS 闪存中隐藏 rootkit 恶意代码并使之在操作系统运行过程中生效。2007 年的 Black Hat 会议上，John Heasman 继续其对固件 rootkit 技术的研究，这一次他将 rootkit 技术应用到了主板 PCI 板卡的 OPROM 闪存中[27]。

美国 2006 年的《计算机安全与信息保障研究与发展联邦计划》(Federal Plan for Cyber Security and Information Assurance Research and Development)[28]提出 12 项计算机安全和信息保障基础技术，其中“硬件和固件安全”名列首位。该文对硬件和固件安全重要性的描述如下(原文引用，第 69 页)[28]：

“Hardware or firmware attacks can undermine even the most sophisticated application-level controls or security mechanisms. Malicious firmware that has unrestricted access to system components (e. g. , if it is part of the OS kernel) has considerable potential to cause harm, introduce backdoor access (an undocumented way of gaining access to a computer, program, or service), install new software, or

modify existing software. If the underlying hardware and firmware cannot be trusted, then the OS and application security mechanisms also cannot be trusted."

国内外近几年越来越多地加速出现一些基于BIOS的应用和安全管理增强产品。Intel公司的"英保通"产品[29]，为企业和网吧用户提供基于硬件和BIOS实现的资产管理、系统分发、磁盘镜像、远程维护等功能，其产品的实现正是基于Intel全新推出的EFI BIOS产品。Phoenix公司也在其传统BIOS产品中增加系统认证、恢复、嵌入式浏览器等功能模块。国内则有中电科技集团30所的基于PCI ROM的磁盘认证和加密保护卡，西安三茗公司的基于BIOS的系统快速恢复和资产管理产品，航天二院706所的基于BIOS的系统用户认证和权限管理产品等。这些产品都是在传统BIOS的层面上，通过附加的嵌入模块利用BIOS增强系统安全管理功能。

1.2.3 固件安全领域新动向

当前信息安全系统的保护主要由防火墙、漏洞扫描、病毒查杀防护等构成，这种砌高墙、堵漏洞、杀病毒的做法只能导致墙越砌越高、漏洞越堵越多、病毒越杀越强的恶性循环。这种恶性循环迫使安全界探索解决计算机和网络安全问题的新思路。其中有两个新的趋势值得重点关注：

（1）**基于固件和硬件的AMT[30]和vPro[31]技术** Intel公司将CPU的vPro技术、EFI固件技术以及以太网控制器技术相结合，为操作系统运行环境构造了一个带外系统。该带外系统构造了一个软硬件过滤器，能够识别网络流量中的恶意代码和潜在攻击，从而在威胁到达操作系统之前将其消灭。为计算机终端构造带外系统能够有效地隔离安全保障者和被保障者，增强安全保障者自身的安全性和免疫性，降低甚至几乎消除安全防护对操作系统和用户工作效率的影响，是信息安全系统防护值得关注的方向。

（2）**TCG和可信计算[32]** 2003年，TCG（Trusted Computing Group，可信计算组织）成立，推动可信计算进入新高潮。TCG的可信计算以可信平台部件TPM为信任基础，以信任链为纽带，以完整性度量和报告为手段，来保障计算终端的可信和安

全。通过终端安全，构建整体的可信计算平台和可信计算生态环境。

产业界是可信计算推动的主导力量，TCG 发起的主体就是 IBM、Intel、HP、微软、Sun 等 IT 企业，成员包括数百家软硬件公司，其规范的制订和推广也是产业界主持和推动的。其中尤其以 TPM 芯片产品进展神速，目前提供 TPM 芯片的厂家包括 Atmel、Broadcom、Infineon、STMicro、Winbond、兆日、联想、武汉瑞达等。我国国家密码管理局也已于 2007 年 12 月发布了《可信计算密码支撑平台功能与接口规范》[33]，其可信密码模块 TCM 同 TPM 标准基本是相对应的。

微软 Vista 操作系统已经提供对 TPM 芯片操作系统和应用软件级的操作和接口支持，可将 TPM 用于对硬盘数据的全盘加密存储技术 BitLocker。国内对于信任链的研究，目前更多地集中于操作系统层信任链的构建工作，包括武汉大学、南开大学、解放军信息工程大学、海军计算技术研究所、北京交通大学、北京工业大学等的一些研究成果[16,34,35,36,37,38]。而对于操作系统下层的 BIOS 层，由于受到技术和试验条件的限制，国内则少有涉及。

产业界大力推动的可信计算，由于完整性度量的局限性，并不能够实现真正的安全，因为这种度量并不能对实体行为进行度量。但 TCG 的可信计算，对于解决信息安全领域目前的困境，还是存在极大的推动作用，不失为解决目前安全困境的一条可尝试的途径。

1.3 本书研究主题与目标

BIOS 在系统安全防护中处于一种基础地位，对于计算终端和操作系统的安全能够起到重要的保护作用。然而，BIOS 安全的重要性在过去很长一段时间内却长期被忽视。伴随 BIOS 安全威胁事件的逐渐增加、安全向硬件和固件延伸的趋势及可信计算的发展，重新审视计算机固件 BIOS 的安全地位，对 BIOS 安全问题进行系统性地研究是非常有必要的。

本书围绕计算机固件安全问题的两个方面系统性地阐述研究工作：

(1) 传统 BIOS 的设计和实现从一开始就没有考虑安全问题,其设计和生产完全是从应用需求出发,导致现有的 BIOS 产品存在较大的安全风险。对现存大规模应用的传统 BIOS 产品所存在的安全漏洞进行分析,研究分析针对 BIOS 系统自身、操作系统和应用所存在的 BIOS 安全威胁及其实现技术,这方面研究成果可应用于基于 BIOS 层的信息安全攻防;研究对计算机 BIOS 系统进行安全检测的理论和方法,开发实现一种计算机 BIOS 安全检测系统,通过安全检测达到维护和增强现有 BIOS 产品及计算机系统的安全性;

(2) 基于新一代 EFI/UEFI 规范,研究新一代安全 BIOS 产品开发的理论和方法。针对可信计算需求,准确定位 BIOS 在可信计算终端中的作用和地位,研究安全 BIOS 系统开发的策略、模型、实现方法和关键技术,开发实现可信 BIOS 的产品原型。

1.4 本书原创性主要贡献

围绕固件系统安全问题的两个方面开展深入研究,本书主要阐述以下五个方面的技术和研究成果:

(1) 以市场现有传统 BIOS 产品为对象,首次比较系统地分析并验证了计算机 BIOS 系统存在的安全漏洞,对漏洞特征和漏洞表示进行研究,建立了第一个计算机 BIOS 漏洞库。

(2) 分析了多起典型 BIOS 安全威胁事件,分析概括了操作系统与 BIOS 系统的依赖性和安全互动性,提出并实现一种新型 BIOS 木马技术。该技术防可用于安全代理,攻可用于嵌入式木马、病毒或 rootkit。

(3) 研究了对计算机 BIOS 进行安全检测的原理和技术方法,提出基于漏洞扫描和恶意代码检测的计算机 BIOS 安全检测模型,实现了一种计算机 BIOS 安全检测系统。该系统能够对目前市场上主流 BIOS 产品进行安全检测,目前已经成功地进行了产品转化。

(4) 通过比较不同历史阶段对可信与安全两个概念的使用情况,阐明可信与安

全两者之间的区别和联系。本书认为,可信的本质是行为的安全,TCG 完整性度量并不总是能够保障系统的安全性,只有行为的可信度量才等价于安全。分析了 BIOS 的安全需求,给出了可信 BIOS 的定义,提出适合 BIOS 系统的可信 BIOS 保护环安全策略和模型。

(5) 通过对 TCG 完整性度量和报告、信任链构建等规范要求的分析,提出基于可信注册、可信封装、可信验证三个步骤的可信 BIOS 可信度量方法;以新一代 EFI/UEFI 规范和 Intel Framework 代码为基础,实现了一个可信 BIOS 的产品原型——UTBIOS。在 UTBIOS 的开发实现中,对可信 BIOS 保护环模型实现的关键问题进行研究,给出了最小 CRTM 构造的约束条件和组成,给出了核心可信代码 CTC 和受保护数据项 PDI 的一个划分实现,设计了 UTBIOS 的可信度量结构。

BIOS 安全的重要性在过去很长一段时间内被忽视。本书这些创新性研究成果对于基于计算机底层固件 BIOS 系统的终端安全防护和可信终端构建的研究,具有积极的推动作用和借鉴意义。

1.5 本书组织结构

本书共分九章,各章内容组织如下:

第 1 章"引言"介绍计算机固件的概念、地位和作用;综述固件安全领域国内外研究现状和趋势,提契本书主要研究成果。

第 2 章"计算机固件的发展与技术基础"考察 BIOS 技术和产品发展历程,比较传统 BIOS 与新一代 BIOS 的优缺点;介绍公用 BIOS 产品、开源 BIOS 项目及我国 BIOS 研发现状;介绍和分析计算机 BIOS 实现的关键技术。

第 3 章"固件安全技术研究开发实例"结合作者对传统固件安全的研究开发成果,以传统固件安全增强和安全代理技术开发为实例,分别阐述了在传统的 Award BIOS 和 Phoenix BIOS 这两种主流的个人计算机和笔记本电脑固件产品中,开发第三方固件安全应用程序的原理、方法和技术手段。

第 4 章“固件 BIOS 安全漏洞及威胁研究”分析 BIOS 安全漏洞和威胁的成因；分析、验证传统 BIOS 产品存在的多种安全漏洞；对 BIOS 安全威胁进行归纳分类，并从技术上重点研究分析了 3 种典型的 BIOS 安全威胁事件；分析 BIOS 与操作系统的互动机制，提出并实现了一种新型的 BIOS 木马(代理)技术。

第 5 章“计算机 BIOS 安全检测方法与实现”分析 BIOS 安全检测的复杂性；通过 BIOS 安全漏洞的验证、特征提取和漏洞表示的系统性研究，首次建立了计算机 BIOS 漏洞库；从理论和技术两方面探讨 BIOS 中恶意代码检测问题，提出一种 BIOS 安全检测模型；完成对主流 BIOS 产品的结构解析，实现了一种 BIOS 安全检测系统。

第 6 章“可信固件开发的策略和模型”辨析了可信与安全两个概念之间的关系及其对信息安全研究的指导意义；通过对 TCG 可信计算技术和规范的分析，明确了固件在可信平台中作为 CRTM 的地位；分析了固件系统的安全需求，通过对经典安全策略和模型的比较分析，提出一种能够满足固件系统安全需求的可信固件保护环模型。

第 7 章“可信度量基础与度量方法”介绍 TCG 可信计算平台的结构、基本特性、可信平台模块、可信根和信任链，分析可信平台中存在的信任关系、证书类型、密钥类型和密钥管理机制；以满足 TCG 可信平台特性、可信度量和报告规范为前提，提出可信固件的可信度量方法。

第 8 章“可信固件的开发实现”介绍本书研究工作开发的可信固件——UTBIOS 实现的软硬件基础；设计并实现 UTBIOS 的 CRTM 结构、可信度量结构与流程；在 UTBIOS 中给出了可信固件保护环模型的一种实现；分析了可信度量对固件系统性能的影响。

第 9 章“结论”总结本书研究工作所取得的主要成果，指出下一步有待继续完善和深入展开的研究工作，提出可开展的进一步研究方向。



第 2 章

计算机固件的发展与技术基础

计算机传统 BIOS 产品和技术的长期垄断，导致国内产业界和学术界对 BIOS 发展历程、开发技术基础等介绍性的文章、书籍等材料较少，这方面稍有影响的书籍只有我国台湾地区旗标出版社出版、陈文钦编写的《BIOS 研发技术剖析》一书，该书一度成为中国大陆可公开参考的传统 BIOS 技术唯一的可参考书籍，也是本书作者初入门时研究 BIOS 技术和 BIOS 安全购买的第一本参考书籍。而且 BIOS 相关的一些专门技术基础也只限定在较小范围的产业专业领域人员所掌握，其技术的公开受到计算机产业链上从芯片厂商、主板厂商到固件厂商的限制。而这些知识对本书后续的研究分析和实现工作的开展又是必不可少的，既是传统 BIOS 安全漏洞和威胁的研究技术基础，又对新一代安全固件 EFI 开发路线的选择具有指导作用。因此，本章结合本书研究过程所获取的一些材料和实践经验，介绍 BIOS 技术、产品发展历程和 BIOS 产品开发的关键基础知识。在不侵犯相关软硬件厂商的知识产权的前提下，按照作者自身研究的体验，尽可能多地介绍固件安全技术研究所需要的固件基础技术资料。

2.1 计算机固件发展历程

2.1.1 传统 BIOS 的演变

当初，Intel 公司设计微型计算机 8086/8088 处理器时，CPU 加电后第一个动作，就是设置代码段寄存器 CS、数据段寄存器 DS、附加段寄存器 ES、堆栈段寄存器 SS 的初始值等于 0FFFFh，设置指令指针寄存器 IP 的值为 0000h，从而将指令指针指向了物理地址 0FFFF0h，从该地址取得第一条 CPU 将要执行的指令。随后 80286、80386、80486、Pentium 等 IA(Intel Architecture)处理器，以及 AMD、VIA/Cyrix 等处理器，也都保留了这样的中央处理器初始化设计特性，以使硬件能够保留这种向后兼容性(Backward Compatibility)[39,40,41]。

最早的 8086/8088 中央处理器只有 20 根地址线，最大寻址空间是 2^{20} bits＝1024KBytes＝1MBytes，十六进制表示就是 00000h～0FFFFFh 这个地址空间。从 CPU 加电令指令指针指向的 0FFFF0h 地址到最大地址空间 0FFFFFh 地址，这之中仅有 16 字节的空间，根本不可能放下任何足够的固件 BIOS 初始化代码段，因此存放在这里的起始指令通常是一条长跳转指令：

```
FFFF:0000 EA5BE000F0 JMP FAR PTR 0F000:E05Bh
```

CPU 加电后，首先从 BIOS 中取得这条长跳转指令，执行这条指令，从而跳转到前端空间的指定位置(0FE05Bh)，从该位置处取得下一条将要执行的指令继续执行。不同厂商和 BIOS 产品类型，长跳转的目标地址也可能有所差别，而原理都是相通的。

中央处理器进化到 32 位后，其寻址空间增加到了 4GB，然而却仍然保留了 CPU 加电后的指令指针设计特性，即 CPU 加电后读取的第一条指令仍然存放在 4GB 空间的最后一个 16 字节的起始位置，物理地址为 0FFFFFFF0h。

需要注意的是，这些固化在计算机主板上芯片中的固件 BIOS 代码的存放区域，

虽然是独立存储在芯片中，占用的却是统一的计算机内存地址区域，通常就是内存的最高端的地址区域，这部分内存区域通常也称为影子内存（shadow）。

第一个创造BIOS的是当时被称为蓝色巨人的IBM公司。IBM公司将最初的固件BIOS资料结构完全公开，使得后来的BIOS产品为保持兼容性都保留了几乎相同的内存区域分配和数据结构。到386平台之前，BIOS固件产品同最初的IBM公司的BIOS在结构、功能、尺寸大小上都没有太大的改变。

计算机进入386处理器平台之后，由于IT业界功能规范的增加，PROM芯片容量的持续增加和价位下跌等因素影响，原来独立存放的CMOS设置代码被加入到BIOS芯片中，再加上BIOS要求能够不断支持各种新增加的外围设备，为这些外围设备提供初始化操作代码、测试操作代码及统一的功能抽象表达层代码，使得固件BIOS的代码容量急剧增加，需要占用0F0000～0FFFFFh一共64KB的空间。而到486平台后，由于中央处理器CPU的多样化，对超大容量硬盘的LBA（Logical Block Addressing）、CHS（Cylinder/Head/Sector）转换支持，新的南北桥集成芯片组的出现，以及PCI总线的流行，都使得BIOS代码容量迅速突破64KB，需要使用128KB的EPROM芯片存储[1]。

伴随着BIOS代码的尺寸增加，为在有限容量的EPROM芯片中能够存放更多的固件代码，BIOS厂商在Pentium处理器时代中期引入了“模块压缩/解压缩”技术。这样，通常BIOS中只有几个不压缩的程序模块（其中包括解压缩模块），而其他模块通常都压缩存储。计算机加电初段，先执行非压缩存储模块的代码，建立起必要的硬件和软件的执行环境，然后调用非压缩存储的解压缩模块代码，对其他将要被执行的压缩模块先进行解压缩操作，然后再执行该模块代码。所使用的压缩算法，往往可使模块代码/数据的压缩率达到30%～60%之间。这种压缩/解压缩技术，使得BIOS厂商能够在寸土寸金的EPROM芯片中，加入功能更多、支持总线、外围设备更齐全的BIOS代码。

到486处理器使用后期，伴随Windows操作系统的出现，对PC的设备智能管理和电源管理功能提出了更高的要求，同时出现了所谓的绿色微机（Green PC）的概念和工业规范，以及高级电源管理（Advanced Power Management，APM）、即插即用

(Plug and Play,PnP)和随后的高级配置与电源接口(Advanced Configuration and Power Interface,ACPI)、USB 总线等功能规范的出现,使得 BIOS 代码容量即使压缩也很难维持在 128KB 之下,通常已经达到 256KB,甚至是 512KB。flash 芯片的出现,允许能够重复擦写 BIOS 中与 PnP 规格相关的数据内容,同时 flash 价格的持续下降,使得固件厂商逐渐转为采用 flash 芯片存储,而容量也扩展为 512KB 甚至是 1MB。

以上所述为传统 BIOS(Legacy BIOS)演变过程,之所以称为传统 BIOS,是为了将其同 Intel 公司在 2003 年发布的遵从扩展固件接口 EFI 规范的新一代固件相区别。从传统 BIOS 的演变过程来分析,本书将传统 BIOS 划分为两代产品。

从 20 世纪 80 年代初期到 20 世纪 90 年代末期这一阶段的 BIOS 可以称为第一代 BIOS 产品。这一代 BIOS 产品的主要技术特点是隐藏硬件平台在芯片组、外围设备控制器等方面的细节上的差别,为操作系统提供统一的硬件平台操作接口,使得操作系统能够在硬件抽象层上执行标准的初始化和操作过程,而能保持在不同硬件平台上良好的兼容性。总之,第一代 BIOS 产品的主要作用,就是要在硬件和操作系统软件之间建立一个方便硬件配置和操作的桥梁,除此之外没有更多的功能需求。

从 20 世纪 90 年代末期之后,EFI 固件之前,这一阶段的固件 BIOS 产品可以称为传统 BIOS 的第二代产品。这一时期的固件 BIOS 产品除了保留第一代 BIOS 的桥梁功能之外,突出的变化是增加了设备智能管理、电源管理以及更多的附属增值应用等,所增加的典型功能包括 PCI 设备的自动枚举、外插卡自动资源分配、计算机休眠/唤醒、计算机远程唤醒、即插即用等。这些新增加到 BIOS 的功能,使得 BIOS 更具有一个小型的微操作系统的本领,即所谓的 Mini OS。甚至于在这个时期,很多笔记本电脑,依托 BIOS 功能,在计算机加电后,无须进入操作系统,就能够为用户提供音乐播放等功能,使用户体验到快捷方便的服务。

2.1.2 传统固件 BIOS 的缺陷

计算机技术,尤其是中央处理器和芯片技术的快速发展,令传统固件 BIOS 背负着沉重的向前兼容负担,小心翼翼向前爬行着发展。固件 BIOS 智能化程度和功能多

样性不断提高，而其运行机制却仍然是建立在20世纪80年代初期的第一代传统固件的技术上，毫无创新和发展。在计算机软硬件技术飞速发展的今天，传统固件BIOS的发展明显缓慢并远远滞后于硬件和操作系统技术的发展，不能充分分享其他软硬件技术发展成果。传统固件BIOS的主要缺陷可以总结如下：

(1) 传统固件BIOS仍然运行在16位实模式下，这使得Pentium 4的CPU在BIOS运行阶段只相当于一块高速的8086中央处理器。并且由于实模式的寻址限制，传统固件BIOS在运行过程中，代码、数据和堆栈也只能限制在1MB的内存空间中，不能充分利用Pentium 4的4GB超大内存空间。尽管后来，在BIOS执行的某些阶段加入了支持4GB的平坦模式的支持，却远远不能充分发挥中央处理器硬件新技术上的优势，令固件BIOS运行阶段的系统可用资源大大受限。

(2) 传统固件BIOS封闭的开发方式严重阻碍了固件BIOS技术发展。传统固件BIOS缺乏统一和开放的架构，不同固件BIOS厂商其BIOS产品结构设计千差万别，并且因为技术保密和市场利益的原因，这些固件BIOS产品的结构对大多数产业链上的厂商来说，都是未知的。芯片厂商需要将新研发的芯片产品交给不同的固件BIOS厂商进行驱动测试，等待该芯片及驱动程序在固件BIOS层工作良好后，才能转交给主板厂商或计算机整机厂商。为了加快这个过程，芯片厂商希望有统一的固件驱动模式和接口，由芯片厂商自行开发出在不同厂商固件BIOS上都能运行测试的驱动，而非由固件BIOS厂商来定制和集成，以加快PC硬件研发的速度。而这种驱动的开发，也希望由传统BIOS开发所采用的汇编语言，转向高级C语言。

(3) 传统固件BIOS的设计，从一开始就缺乏安全方面的考虑。

2.1.3　新一代固件EFI/UEFI

微处理器厂商Intel公司很早就意识到传统固件BIOS技术上的缺陷。该公司从1997年起，较早就从公司内部悄然发起对传统固件BIOS技术的革新。1999年，Intel公司开发出“Intel平台创新框架”(Intel Platform Innovation Framework)[42]作为新一代系统固件开发的体系架构。在这个体系架构下，Intel公司定义了扩展固件接口

(Extensible Firmware Interface,EFI)[5]规范,规范中规定了固件中硬件驱动程序以及固件同操作系统接口的新标准,并首先在 64 位 Itanium 服务器上使用了这种创新的固件产品,取得了良好的效果。随后在 2003 年,Intel 公司正式向外界推出,将符合 EFI 1.10 规范的平台创新框架产品移植到 IA 32 位处理器上,并在 Intel 公司的个人计算机主板产品上全面采用符合 EFI 规范的新一代 BIOS 产品。为推动这种新固件架构的发展和广泛使用,Intel 公司开源了一个称为 Tiano 的 EFI 1.10 规范的固件实现框架的源代码,并在 2005 年联合 PC 业界其他厂商一起,成立了 UEFI(Unified EFI)论坛,致力于开发业界公认的 EFI 固件标准。UEFI 委员会以 EFI 1.10 规范为基础,于 2006 年发布了 UEFI 2.0 标准[6],2007 年 1 月发布了 UEFI 2.1 标准[7]。截至本书写作为止,目前 UEFI 标准的最新版本是 2.3.1。

Intel 平台创新框架和 EFI、UEFI 规范的成功创立,解决了传统固件 BIOS 在运行机制方面的缺陷,为业界提供了统一的固件驱动和应用程序模式。它定义了 EFI 驱动程序(EFI Driver)和 EFI 应用程序(EFI Application)结构,允许硬件开发商、操作系统开发商、应用软件开发商以及普通用户从外部存储设备中加载各自开发的硬件驱动程序和固件应用程序。另外,还提供了固件层次的程序调试诊断工具和调试运行仿真环境;定义了统一的引导服务(Boot Services)和运行时服务(Runtime Services)以满足固件和操作系统两个层次的功能调用需求;并且它在运行阶段,很早就从中央处理器加电默认的实模式切换到 32 位保护模式下运行。这种创新的固件核心架构和技术,正是计算机工业界所一直希望得到的结果,因此得到业界各层次厂商的迅速支持。UEFI 规范在短短几年中,迅速发展成为新一代固件不可避免的主流和事实上的业界标准。

遗憾的是,这种新一代固件创新架构在增加了标准性、先进性、互动性的同时,并没有解决传统固件存在的固有的安全问题,反而是为固件层次带来了更多的安全问题。当然,在最初的设计中,Intel 公司已经为这种新一代固件平台创新架构预留了部分安全设计的接口[42]。不过 Intel 公司在其新一代固件产品实现中,这些接口的实现仍然是一个空白,只是给出了指导性的实现建议,而更多具体的安全功能,则需要固件厂商或者是其他第三方软硬件厂商去实现。而本书研究结果也表

明，这些安全接口的定义和实现，也远远不能满足固件层次越来越复杂的安全运行需求。

值得一提的是，介于传统固件和新一代 EFI 固件之间，还出现了一种固件雏形较有代表意义，而最终在同 EFI 规范的竞争中败下阵来，成为过眼云烟。这就是全球最大的固件 BIOS 厂商美国 Phoenix 公司在 2003 年与 EFI 同期提出的核心系统软件固件 CSS BIOS(Core System Software BIOS)。Phoenix 公司提出的这种新型固件计划同微软公司合作，将固件 BIOS 直接嵌入到微软公司的下一代操作系统中，使固件成为操作系统的一部分。微软公司也长久以来就一直希望能够跳过固件 BIOS 直接控制计算机硬件，但却从来没有真正地成功过。通过直接被放入到 Windows 操作系统最底层中的 CSS BIOS，上层图形界面用户就可以通过 CSS 技术直接控制计算机硬件，换个角度理解就是操作系统可以直接访问和管理所有硬件资源，从而彻底抛弃技术已经落后过时的传统 BIOS 固件。

从技术和功能角度来看，CSS 在操作系统与硬件高度结合的优势下，对比传统 BIOS 具有数字安全、网络管理、灾难恢复等方面的优点。从用户的角度来看，CSS BIOS 的最大好处在于将计算机操作变得更加简单可靠，具有更强的可维护性，配置管理界面都有大幅度的改善。这样的固件 BIOS 或者只是 Windows 中的一个组件，只需要打开控制面板做出相应的调整就可以简单明了地调整硬件设置，并且即时生效。这听起来似乎十分理想，不过 CSS BIOS 固件同样存在很多缺点。首先，CSS BIOS 只是把放在 flash 芯片等非易失性存储体中的固件，换成纯软件的形式存储在操作系统中，对传统固件 BIOS 系统本质上的变革有限。其次，若操作系统直接包含固件 BIOS 固然可强化对硬件的控制能力，但如果恶意程序取得对操作系统的控制权，就能直接对系统安全做出更具破坏力的操作，或更容易对计算机用户的操作进行监视、窃取资料。外部程序侵入成为 CSS BIOS 固件最直接的隐患，一旦外部程序取得计算机的控制权，那么 CSS BIOS 将会变成最大的敌人，在连 UNIX 都常被攻破的今天，漏洞百出的 Windows 如何保证自身的安全呢？

CSS BIOS 技术看起来很诱人，不过，目前符合 EFI 规范的固件已经成为事实上的新一代固件工业标准，发展迅速，而同期提出的 CSS BIOS 此后一段时间却无任何

发展，其提出者 Phoenix 公司和微软公司，都已经在 2007 年开始加入 UEFI 阵营，转而支持 UEFI 标准。两种新一代固件架构和技术的优劣之争显而易见，以 Intel 公司的 EFI 最后胜出而告终。

以 UEFI 为标准的新一代固件，可以称为第三代固件。第三代固件同第一代和第二代传统固件 BIOS 相比较，不再是简单的继承、修改和完善，而是一次彻底的创新。表 2.1 对传统固件 BIOS 和新一代 EFI 固件从差异性方面作了比较。

表 2.1　传统 BIOS 与新一代 BIOS 差异特性比较

序　号	特　性	传统 BIOS	新一代 EFI BIOS
1	运行模式	16 位实模式	32 位保护模式
2	用户界面	文本界面	图形界面
3	系统开发语言	汇编语言	C 语言(少量汇编)
4	驱动开发语言	汇编语言	C 语言，bytecode
5	TCP/IP 协议支持	不支持	支持
6	BIOS 层调试运行环境	无	EFI Shell
7	外部驱动加载、标准	不支持	支持、EFI Driver
8	外部应用加载、标准	不支持	支持、EFI Application

2.2　固件产品和技术研发状态

2.2.1　公用固件产品

计算机传统固件 BIOS 产品市场和研发技术一直以来被我国台湾地区和美国的公司所垄断。由于牵扯到专利权与厂商之间的竞争因素，固件 BIOS 产品的兼容性与合法性，一直是试图进入这一领域的业界企业要面对的严肃问题。1983 年，“台湾工研院”电子工业研究所(ERSO)，以纯净室(Clean Room)方式，开发出与 IBM PC 相兼容的 ERSO BIOS，并授权转移给我国台湾地区厂商使用。成立于 1979 年的美国厂

商 Phoenix Technologies 公司以同样的纯净室方式，在 286 时代开发出功能强大且兼容的 286 AT BIOS。随后，AMI(American Megatrends Inc.)、Award Software Inc. 等也相继加入固件 BIOS 产品的角逐，形成了美国厂商 Phoenix 公司、AMI 公司和我国台湾地区厂商 Award 公司在商业固件 BIOS 市场上三足鼎立的局面[1]。

1998 年 9 月美国 Phoenix 公司收购 Award 公司。而同期(1998 年 9 月)成立的“台湾系微股份公司”(Insyde)买下 SystemSoft 公司的固件 BIOS 版权与相关研发部门，并同 Intel 公司合作，开拓传统固件 BIOS 和 EFI 固件产品研发和市场。由此，垄断固件设计生产的仍然是我国台湾地区的 Insyde 公司、美国的 Phoenix 公司和美国的 AMI 三家公司。

2.2.2　开源固件 BIOS 项目

为打破传统固件 BIOS 技术的垄断和封锁，国际上出现了一些固件 BIOS 开发和设计的开源项目和组织。这其中比较有影响的有 TinyBIOS 项目、OpenBIOS 项目和 LinuxBIOS 项目。但由于固件 BIOS 技术与芯片组和平台密切相关，这些开源 BIOS 项目由于难以建立同产业链上的硬件厂商的联合，因此其项目固件支持的芯片组和平台非常有限，只能在少数特定硬件平台上运行，而且其固件的实现对现有工业标准的支持(PnP、ACPI 等)也并不全面。不过这些开源固件 BIOS 项目的出现，为打破传统固件 BIOS 技术的垄断，不断推进固件 BIOS 技术的开放和发展还是作出了重要的贡献。

1. TinyBIOS

TinyBIOS(http://www.pcengines.ch/tinybios.htm)是一个为嵌入式 PC 开发的开源固件 BIOS 项目，固件 BIOS 中同嵌入式需求不相关的部分都未被实现，如电源管理、BIOS 设置程序(BIOS Setup)等。目前该项目支持的操作系统包括 DOS、Linux、FreeBSD、Windows NT。开源的 TinyBIOS 只支持两种芯片组：ALI(Acer Labs) M1487 FINALi 和 ALI M6117，其他芯片组的支持则需要购买授权或由用户

自行实现。从架构和实现技术上看，TinyBIOS实际上是传统BIOS的缩减版。

2. OpenBIOS

OpenBIOS(http://openbios.info/Welcome_to_OpenBIOS)也是一个开源BIOS项目，其目标是开发免费、开放源代码、不依赖于任何硬件架构的固件BIOS，为传统商业固件BIOS提供一个替代产品。OpenBIOS的开发遵循IEEE 1275-1994的标准[43](通常称为Open Firmware标准)，并且设计了一种FCode中间语言，使得OpenBIOS的代码实现在编译后不依赖于任何一种CPU指令集，使得代码具有很好的不同硬件平台的兼容移植性。OpenBIOS的目标是能够支持各种不同的硬件平台，包括如X86、AMD64、PowerPC、ARM和Mips等各种服务器、工作站和嵌入式设备。在SUN、Apple、IBM的某些服务器和工作站中已经应用了OpenBIOS某些固件产品。OpenBIOS优先支持的是UNIX和Linux操作系统，对微软Windows操作系统的兼容性支持有待进一步完善。

大多数情况下，OpenBIOS依赖于更底层的固件来完成平台硬件的初始化工作，如OpenBIOS常常同LinuxBIOS、U-Boot一起使用。从这个意义上讲，OpenBIOS也只是完成传统商业BIOS产品的部分上层工作，不是真正意义的完整BIOS固件。

3. LinuxBIOS

LinuxBIOS(http://www.coreboot.org/Welcome_to_coreboot)同样是一个开源的自由软件项目。LinuxBIOS项目的技术思路很有创意，它采用相对较少的代码来初始化平台硬件设备，然后快速进入一个称为payload的核心执行层，这种核心执行层可以是如Linux Kernel、Grub2、OpenBIOS等，由核心执行层来完成更多更复杂的平台设置和用户应用需求工作[8]。就像LinuxBIOS的名字显示的一样，这个项目最初是为了支持Linux内核的快速引导，不过随着LinuxBIOS的发展，目前已经开发出称为ADLO的payload，用于在LinuxBIOS层上提供对Windows系统引导的支持。ADLO项目源于开源的Bochs(http://bochs.sourceforge.net/)项目，其提供了对16

位传统 BIOS 的虚拟实现支持。

LinuxBIOS 运行过程很早就切换进入 32 位保护模式，基本采用 C 语言进行开发，这同 Intel 的 EFI 固件很相似。

LinuxBIOS 只有同其上的 payload 一起工作，才能完成传统固件 BIOS 的功能。但其上的 payload 又可以进行扩展，甚至等同于一个微型操作系统(Mini OS)，将 LinuxBIOS 和这个 Mini OS 一起封装进芯片中，成为一个界面丰富的嵌入式系统。在嵌入式领域，LinuxBIOS 具备一定优势。

4. Bochs

Bochs(http://bochs.sourceforge.net/)实际上不是一种固件，而是一种十分轻便的使用 C++编写的开源 IA-32(x86)计算机模拟器，可以运行在多种平台上。它仿真英特尔 x86 CPU、常见的 I/O 设备和定制的固件 BIOS。目前，Bochs 可以被编译仿真 386、486、Pentium/Pentium Ⅱ/Pentium Ⅲ/Pentium 4 或 x86-64 位的 CPU，包括可选的 MMX、SSEx 和 3DNow！指令。在 Bochs 仿真环境里能够运行许多操作系统，如 Linux、DOS、Windows 95/98/NT/2000/XP 或者 Windows Vista。Bochs 可以被编译运用在多种模式下，其中有些仍处于发展中。

之所以在开源固件项目中介绍 Bochs，是因为在 Bochs 项目的开源代码中，有使用 C/C++代码编写的较完整的传统固件 BIOS 的仿真实现代码。阅读这些代码，毫无疑问能够更深入地了解和掌握传统固件 BIOS 技术。甚至于一些厂商，通过修改使用 Bochs 中的固件 BIOS 仿真实现代码，来实现 EFI 固件中要求的兼容性支持模块(Compatibility Support Module，CSM)[52]，以完成 EFI 固件对要求使用传统固件 BIOS 功能和接口的旧有操作系统的兼容支持。

2.2.3 我国计算机固件产品研发现状

我国信息技术产业在计算机固件领域长期以来处于空白局面。联想、方正、长城、浪潮等国内计算机或主板大厂商，其计算机或主板上固件 BIOS 都需要向专业固

件 BIOS 厂商取得授权，并请固件 BIOS 厂商作必要的技术协助，针对计算机硬件平台特性作必要的修改和移植工作。

固件 BIOS 作为信息产业链的一个重要环节，其核心技术一直由我国台湾地区和美国少数几个企业垄断，这不仅影响我国建立完整的产业链，对国家信息安全来讲也是一个不可忽视的隐患。国内一直在试图寻求国产自主固件 BIOS 产品和技术突破，开源 BIOS 项目为这种突破提供了机会，而 Intel 公司新一代 EFI 固件技术推动了固件产业界的重新洗牌、整合和利益分配，是我国企业切入固件市场千载难逢的机遇。2005 年，在信息产业部（现工业和信息化部）参与和支持下，由 Intel 公司提供核心技术，在国内扶植一到两家新一代固件厂商。与此同时，信息产业部通过电子产业发展基金支持，启动“新一代安全 BIOS 的研制和产业化”、“高安全与可管理 BIOS”等项目，开展自主研发适合我国国情的计算机固件产品和技术的研究工作。

2.3 固件开发基础技术与规范

固件技术同处理器、芯片组、主板等硬件技术密切相关，又由于多年的传递和继承，其向前兼容性要求较多，而且由于固件处于硬件和操作系统软件的中间层，因而需要遵守许多软硬件接口的规范。因此，固件的开发是一项琐碎麻烦的工作。本节关注于传统固件 BIOS 开发的硬件以及相关兼容性、规范性技术介绍，特别是同固件安全技术分析和开发息息相关的那些固件技术环节。同时，也对 UEFI 固件的相关规范和技术环节作了概括的介绍。至于详细的固件开发技术，传统固件推荐读者研读参考文献[1]，而 UEFI 固件则需要耐心研读 UEFI 官方网站提供的资料，不作为本书研究介绍的重点。而至于传统固件框架性结构则是各个固件 BIOS 厂商各不相同，可以自定的，在此不作统一介绍，后续章节设计固件安全技术时，再按照需要做阐述。这些基础技术与规范对本书后续的固件安全问题的研究是必不可少的知识基础。

2.3.1 硬件体系架构

固件的开发同硬件平台密切相关，开发者必须对固件所依赖的硬件体系结构、不同平台的主板结构和南北桥芯片组的细微区别有清楚的了解，才能使固件对所依赖的硬件平台能有全面准确的支持。

以本书在 EFI 固件研究实验中所使用的 Intel D945gnt 主板为例，CPU 使用 LGA775 socket 的 Pentium 4 芯片，北桥芯片(North Bridge)采用 Intel 82945G 芯片组[44]，南桥芯片(South Bridge)采用 Intel 82801G 芯片组[45]。如图 2.1 所示是 Intel 945G 南北桥芯片组结构示意图[44]。由图可以看出，北桥芯片负责内存、显卡、CPU 之间的集成或连接，而南桥芯片负责 PCI、IDE、USB、SATA 存储设备以及键盘、鼠标、网卡等外围接口的集成或连接处理。存储固件的芯片通过 SPI(Serial Peripheral Interface)接口或 LPC(Low Pin Count)接口连接在南桥芯片上。

本书研发实验工作中所使用的主板 Intel D945gnt，其 BIOS flash 是通过 SPI 接口连接在南桥芯片 ICH7 上。另外，该款主板预留有可信平台模块 TPM(Trusted Platform Module)插槽，通过 LPC 同 ICH7 相连。可信平台模块 TPM 能够为加密和数字签名密钥提供基于硬件的保护。如图 2.2 所示为 Intel D945gnt 主板的功能性结构示意图[46]。

在固件设计开发中，同硬件体系结构密切相关的、至关重要的前期初始化主要包括三部分：中央处理器 CPU 初始化、内存初始化、系统芯片组初始化。其他外围控制器的初始化工作则可以在后续运行过程中根据需要逐一进行操作。下面就这三部分初始化工作的关键点作部分阐述。

1. 中央处理器 CPU 初始化[39,40,41]

IA32 处理器可以工作于 3 种模式：实地址模式(Real-address)、保护模式(Protected)以及虚拟 8086 模式(Virtual 8086)。实地址模式提供了 Intel 8086 处理器的运行环境，虚拟 8086 模式可让处理器在保护模式下可运行 8086 模式软件及多

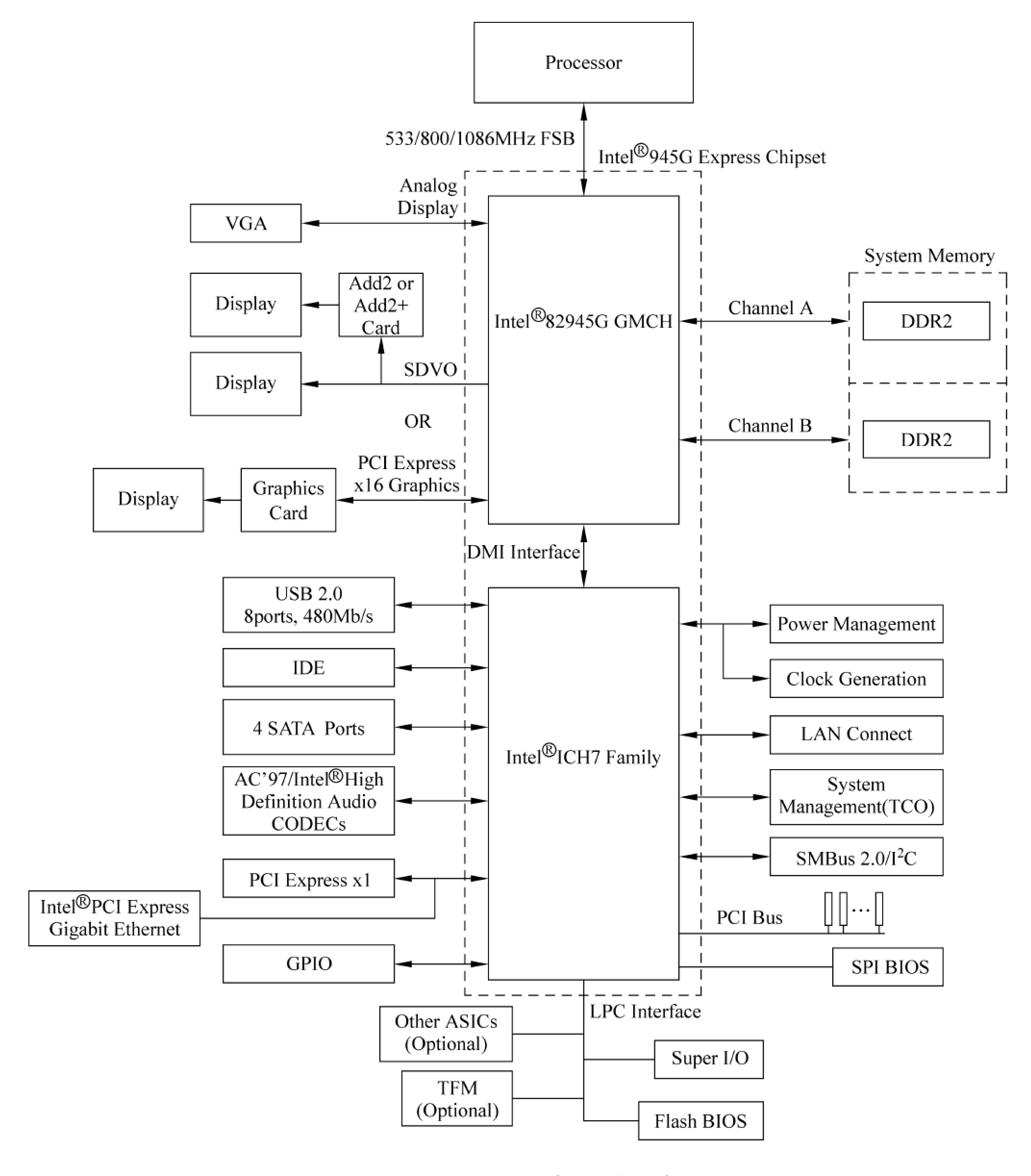


图 2.1 Intel 945G 芯片组结构示意图

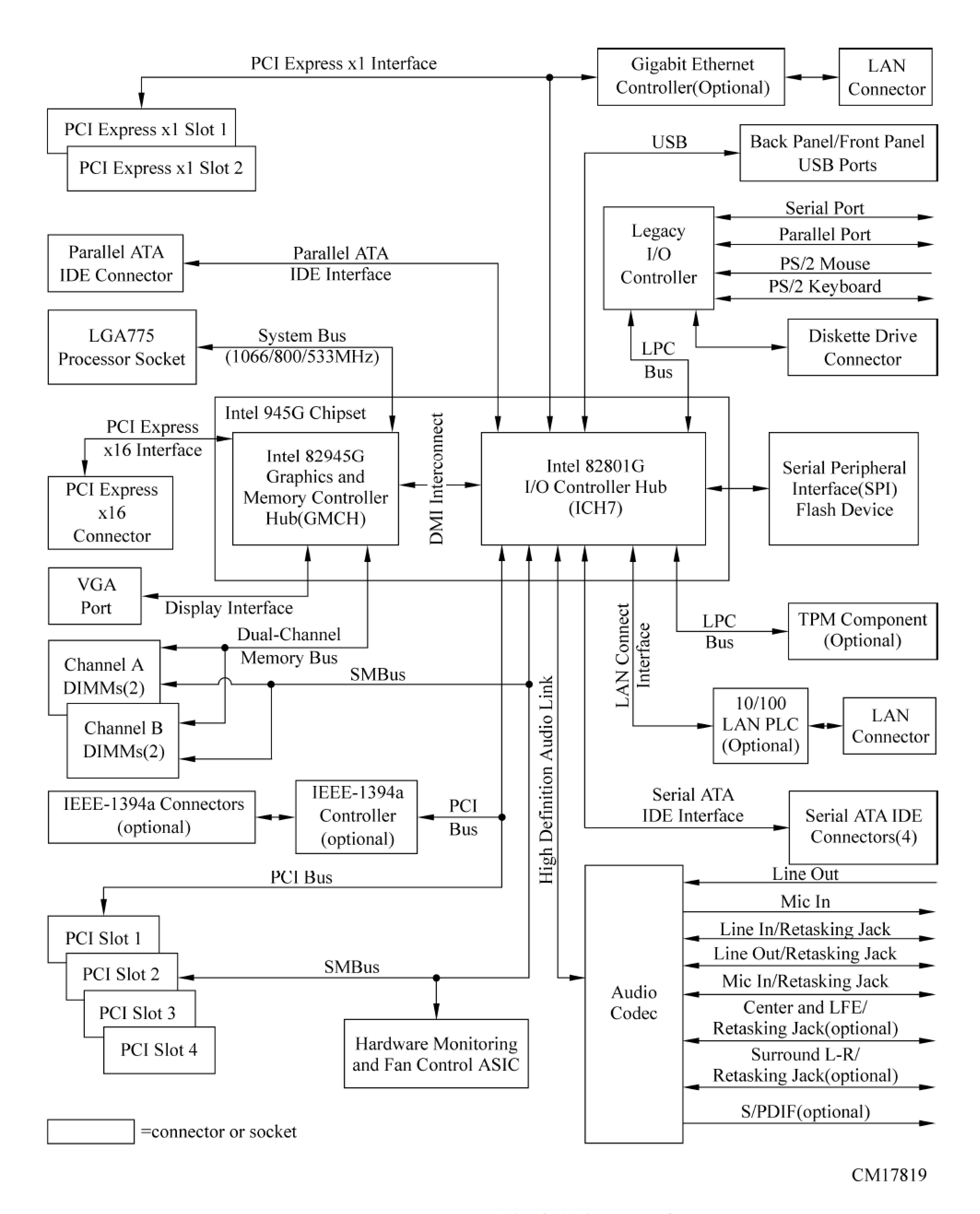


图 2.2　Intel D945gnt 主板功能性结构示意图

任务环境，保护模式则是32位处理器本身的操作模式，在这种模式下，所有的指令和特定结构能发挥最高的性能和兼容性。

IA32系列处理器加电后都自动运行于实地址模式下。第一代和第二代固件BIOS产品，在固件BIOS获取系统控制权后，保持处理器一直运行在实地址模式下，直到操作系统加载后，由操作系统将处理器转换到保护模式下运行。即第一代和第二代固件BIOS，不管处理器类型如何，其所有代码都运行在16位的实地址模式下，没有能够随着处理器的升级换代而采用高效安全的保护模式。第三代EFI固件产品同前两代传统固件产品相比较，一个突出的要求就是在处理器加电运行后，尽量早地将处理器从实模式转换到保护模式，使得绝大部分固件代码可以运行在高效安全的保护模式下，充分发挥中央处理器的性能。

固件对中央处理器CPU的初始化，往往还包括MP(Multi-Processor，多处理器)特性、HT(Hyper-Thread，超线程)特性以及CPU一级缓存(L1 Cache)和二级缓存(L2 Cache)的处理。

2. 内存初始化[39,45,46]

固件代码对CPU初始化完成后，接下来需要对系统内存进行初始化，使得系统有可用的RAM内存。DRAM构造复杂，要正常运行需要进行相当繁琐的参数设置。

DRAM主要相关参数如下。

(1) RAS-to-CAS Delay：DRAM中RAS到CAS延迟时间，设定不当可能会造成内存存取错误。所谓RAS(Row-access strobe，行地址触发)是内存寻址的前段动作，而CAS(Column-access strobe，列地址触发)则是后段部分。由于总线复用的缘故，所以在RAS寻址或CAS寻址时需要进行解码和锁定。但是在行解码之时，由于解码与锁定两个动作必须花去一部分时间，如果中间完全没有延迟就开始列解码动作，那么可能会由于动作的延迟导致存取错误以致死机，所以需要进行RAS与CAS之间延迟时间的设定。

(2) RAS Precharge Time：代表当RAS(行地址触发)在DRAM做重置动作之前所预留的充电时间，一般设定为2ns或3ns。预充电时间如果不足，可能造成内存无

法正确重置而导致数据丢失。因为 DRAM 在设计上是采用刷新电容方式存储数据，电容中电量会随时间流失而不断削减甚至消失，从而造成数据流失。为了维持高效又避免数据流失，因此动态内存必须不断进行充电，以维持其中的数据。

（3）CAS Latency Time：DRAM 读取或写入动作的 CAS 触发信号前的延迟时间。

（4）DRAM Data Integrity Mode：一部分 DRAM 支持 ECC（Error Checking and Correction）功能。大多数内存并不支持 ECC 校验。

（5）Host/DRAM Frequency：DRAM 运行频率，如设置为 66MHz。

（6）DRAM Row Boundary Registers：简称 DRB 寄存器，用于计算 RAM 大小。共有 8 个寄存器，各寄存器中数据以 8MB 为单位，分别读取之，并按如下方式进行计算：

DRB0 ＝ Total memory in row0（in 8MB）

DRB1 ＝ Total memory in row0＋row1（in 8MB）

DRB2 ＝ Total memory in row0＋row1＋row2（in 8MB）

DRB3 ＝ Total memory in row0＋row1＋row2＋row3（in 8MB）

DRB4 ＝ Total memory in row0＋row1＋row2＋row3＋row4（in 8MB）

DRB5 ＝ Total memory in row0＋row1＋row2＋row3＋row4＋row5（in 8MB）

DRB6 ＝ Total memory in row0＋row1＋row2＋row3＋row4＋row5＋row6（in 8MB）

DRB7 ＝ Total memory in row0＋row1＋row2＋row3＋row4＋row5＋row6＋row7（in 8MB）

这样即可获知 DRAM 内存的容量大小。

那么固件要进行设置的这些参数的正确数据从何而来？它们都被存储在内存条正面右侧的一个 8 针的 256 字节的 EEPROM 芯片里。这个芯片被称为 SPD（Serial Presence Detect，串行存在探测），里面记录着内存的速度、容量、电压与行、列地址带宽等参数信息。初始化内存复杂的地方就在于进行 DRAM 设置前，必须要通过 SPD 来获取相关的信息，然后按照其中的数据对 DRAM 控制器进行参数设置。而读取

SPD，则是通过南桥上的SMBUS总线来进行。不同芯片组和内存的组合，其设置需要开发者细读相关厂家的Data Sheet来了解。

3. 系统芯片组初始化[45,46]

早期主板上，各种组件之间的连接与协调电路极为复杂，动辄使用到上百块各类型的IC、电容，不但占用主板面积，也造成了除错上的麻烦与无谓的成本。随着半导体科技的进步，主板上的许多线路，已使用几个大型IC芯片来取代，可以简化设计与除错时间、减少成本，这些专攻主板简化设计的芯片，就称为系统芯片组[1]。一般主板上会有几个大型的芯片，每一个芯片有着特定的功能，这其中最重要的就是南北桥芯片组。以Intel D945gnt主板为例，离CPU最近的芯片是Intel 82945G芯片，称之为北桥芯片(North bridge)，主要负责内存和显示控制；另一个是Intel 82801G芯片，称之为南桥芯片(South bridge)，主要负责外围的输入/输出控制。

系统芯片组初始化是极其复杂的一项工作，往往需要芯片组厂商提供详细的Data Sheet甚至初始化代码来完成。芯片组厂商对于新系统芯片组的这部分资料往往要作为商业秘密，只在密切相关的小范围内授权公开，这也是造成30年来固件BIOS技术高度垄断的一个重要原因。

2.3.2 总线接口规范

任何一个微处理器都要与一定数量的部件和外围设备连接，但如果将各部件和每一种外围设备都分别用一组线路与CPU直接连接，那么连线将会错综复杂，甚至难以实现。为了简化硬件电路设计、简化系统结构，常用一组线路，配置以适当的接口电路，与各部件和外围设备连接，这组共用的连接线路被称为总线。采用总线结构便于部件和设备的扩充，尤其制定了统一的总线标准使不同设备间容易实现互连。

微机中总线一般有内部总线、系统总线和外部总线。内部总线是微机内部各外围芯片与处理器之间的总线，用于芯片一级的互连；系统总线是微机中各插件板与系统板之间的总线，用于插件板一级的互连；外部总线则是微机和外部设备之间的

总线，它用于设备一级的互连。

计算机固件要通过这些总线发现和访问主板上连接的芯片和设备，对这些芯片或设备的控制器(Controller)进行配置，并负责把设备初始化配置、数据结构、驱动ROM等信息传递给操作系统。固件开发者必须清楚如何操控这些总线去完成芯片和设备的发现和初始化工作。本节对同计算机固件开发密切相关的几种总线作一简单介绍。

1. LPC 总线

LPC(Low Pin Count)总线是Intel公司于1997年公布的一个取代传统ISA总线的一种新接口规范。Intel所定义的LPC接口，将ISA总线的地址/数据分开译码改成类似PCI的地址/数据线共享的译码方式，信号线数量大幅降低，工作速率由LPC总线速率同步驱动。改造后的LPC一样维持同ISA相同的16MB/s的最大传输值，但所需要的信号线大幅降低至25～30条。采用LPC总线接口的芯片管脚数和体积都能够大幅度缩减，主板的设计也可以简化。主板上的Super I/O芯片、Flash ROM芯片以及TPM芯片通常通过LPC总线同南桥芯片相连接。

2. SMBUS 总线

SMBUS(系统管理总线)是系统组件管理总线，该总线接口拥有两组信号线，属于低速率(80～400kHz)接口，供符合SMBUS协议的外围组件检测、定位、读写参数等设置使用。一般主板由南桥芯片内置SMBUS控制器(SMBUS Controller)，可通过它来检测DRAM插了几排，并自动抓取该排DRAM时序参数，以及读取硬件监控芯片参数来控制CPU、主板工作温度、工作电压、风扇转速等。前面提到关于DRAM控制器的设置就是通过SMBUS总线读取SPD中内容来进行的。

3. ATA 总线

ATA(Advanced Technology Atachment)接口标准最初是在1986年由COMPAQ和Western Digital两家公司共同开发的，是专用于硬盘的接口总线标准。

在 ATA 接口标准的整个发展过程中，可以划分为 7 个不同的版本。

ATA-1：第一代的 ATA 标准，即 IDE（Integrated Drive Electronics）标准。ATA-1 只支持 PIO-0（Programmed IO）、PIO-1 和 PIO-2 模式，其数据传输速度只有 3.3MB/s，根据其技术标准，硬盘容量限制在 504MB 之内。

ATA-2：也就是常说的 EIDE（Enhanced IDE）或 Fast ATA，它在 ATA 的基础上增加了 PIO-3 模式，不仅将硬盘的最高传输率提高到 16.6MB/s，还同时引进 LBA 地址转换方式，突破了固有的 504MB 的限制，可以支持最高达 8.4GB 的硬盘。

ATA-3：ATA-3 并没有提高 IDE 接口的工作速度，最高传输速度仍为 16.6MB/s（支持 PIO-3），但引入了密码保护机制，对电源管理方案进行了修改，引入了 S.M.A.R.T（Self-Monitoring Analysis and Reporting Technology，硬盘自监测、自分析和报告技术）。

ATA-4：自这一版本开始，硬盘开始支持 DMA 技术，所以又称之为 Ultra ATA/33。硬盘控制器采用总线主控方式进行数据传输，它将 PIO 下的最大数据传输率提高了一倍，达到 33MB/s，也称为 PIO-4。

ATA-5：这一版本就是市面上标注为 Ultra ATA/66 的硬盘，不仅将接口通道的数据交换速度提高了一倍，同时也继承了上一代 Ultra ATA/33 的核心技术——冗余校验技术（CRC）。

ATA-6：这就是市面上标注为 Ultra ATA/100 的硬盘接口标准。这一标准主要是提高了硬盘数据的传输速率，从原来 ATA-5 标准中的 66MB/s 提高到 100MB/s。

ATA-7：较新的硬盘接口标准，传输速率达到了 133MB/s。

上述 ATA-1 到 ATA-7 均是采用并行数据传输方式的总线接口标准，通常也称为 PATA（Parallel ATA）总线接口。最新一代的硬盘接口总线标准 SATA（Serial ATA）采用串行方式传输数据。SATA 是一种完全不同于 PATA 的新型硬盘接口类型，与 PATA 相比，SATA 具有比较大的优势。首先，SATA 以连续串行的方式传送数据，可以在较少的位宽下使用较高的工作频率来提高数据传输的带宽。SATA 一次只会传送 1 位数据，这样能减少 SATA 接口的针脚数目，使连接电缆数目变少，效

率也会更高。实际上，SATA 仅用四支针脚就能完成所有的工作，分别用于连接电缆、连接地线、发送数据和接收数据，同时这样的架构还能降低系统能耗和减小系统复杂性。其次，SATA 的起点更高、发展潜力更大，SATA 1.0 定义的数据传输率可达 150MB/s，这比目前最快的 PATA（即 ATA/133）所能达到的最高数据传输率 133MB/s 还高，而目前 SATA Ⅱ 的数据传输率则已经高达 300MB/s。

固件需要按照 ATA 标准对硬盘进行配置操作，发送和接收命令/数据/响应，以便读取硬盘数据，引导操作系统。

4. USB 总线

USB（Universal Serial Bus）总线提供中低速率外围设备的扩充能力，这些设备包括 USB 键盘、鼠标、移动存储卡、移动 flash 盘等。USB 接口允许真正的热插拔操作，达到即插即用的理想效果。USB v1.1 提供最大 1.5MB/s 的传输速率，而 USB v2.0 传输速率则达到 60MB/s。

由于 USB 驱动方式比较复杂，一般 USB 设备都需要进入操作系统下，加载适当的驱动程序来驱动，或由操作系统提供统一兼容的驱动，而在固件 BIOS 运行阶段一般不支持 USB 设备的操作。但随着 USB 键盘、鼠标、硬盘、光驱的越来越广泛的使用，在固件 BIOS 中提供对 USB 的支持，尤其是 USB 键盘、鼠标的支持已不可避免。Phoenix 公司等传统固件 BIOS 厂商已经在其最新的 BIOS 产品中提供了对 USB 键盘、鼠标、光驱、硬盘、U 盘的驱动和读写支持，EFI/UEFI 固件规范更是将对 USB 的驱动和支持以标准协议（Protocol）的形式写入规范中。

5. PCI 总线

PCI（Peripheral Component Interconnect）是一种先进的第二代局部总线，用来替换原有的低速的第一代 ISA（Industry Standard Architecture）、EISA（Extended ISA）、VESA（Video Electronics Standards Association）总线。PCI 总线目前已经成为使用最多最广泛的计算机标准总线。32 位的 PCI 33MHz 和 PCI 66MHz 总线峰值带宽分别为 133MB/s 和 266MB/s，而改良的 32 位 PCI-X 533MHz 峰值带宽达

213MB/s。最新的第三代高性能总线 PCI Express 单通道 x1 的总带宽达到 512MB/s，而随通道数的翻倍，其总带宽也迅速翻倍，x16 的总带宽则达到 8GB/s。

无论是传统固件 BIOS 还是新一代的 EFI 固件，对 PCI 总线的处理都是一项重要的工作。固件工程师需要熟悉 PCI 的规范，清楚了解 PCI 的配置空间和工作原理，正确处理 PCI 设备上可能附加的 PCI ROM 固件，才能让固件 BIOS 正确配置驱动连接在 PCI 总线上的各种设备。

固件中对 PCI 总线的处理包含两部分重要工作：

(1) PCI 总线配置

驻留在 PCI 总线上的设备包含一种或多种功能(包含多种功能的设备称为多功能设备)。一台设备最多可能包含 8 种功能，编号为 0～7。每种功能各自负责实现它的一组配置寄存器，通过访问这些寄存器可以发现某种功能是否存在，以及对它进行配置，以便进行正常的操作。除了内存、I/O 和消息空间外，PCI Express 还定义有一块专用的配置空间，该配置空间要分配给每一种功能，以便实现它的配置寄存器。关于 PCI 配置空间的详细描述参见 PCI 规范文档[47,48]。

固件中对 PCI 的配置可以采用两种不同的配置机制：PCI 兼容配置机制和 PCI Express 增强配置机制。

PCI 兼容配置机制通过 32 位 I/O 端口 0CF8h(Configuration Address Port)和 0CFCh(Configuration Data Port)访问 PCI 兼容配置寄存器[44,49]。访问步骤为：将目标总线号(bus number)、设备号(device number)、功能号(function number)及双字数(dword number)写入配置地址端口 0CF8h，并将其中的允许位设置成 1，然后按字节、字或双字方式从配置数据端口读取或写入配置寄存器数据。

图 2.3 显示了配置地址端口地址内容的格式。

PCI Express 增强配置机制为每种 PCI 功能的 4KB 配置空间在 256MB 基址对齐的存储空间中都有一个相对应的起始地址，从这个地址开始连续 4KB 的区域将被用作配置空间。通过对这部分内存空间的直接访问实现对 PCI Express 配置寄存器进行操作，这种方式也称内存映射的 I/O 读写(Memory-mapped I/O R/W)。表 2.2 给出增强配置机制存储器映射的 I/O 地址格式。

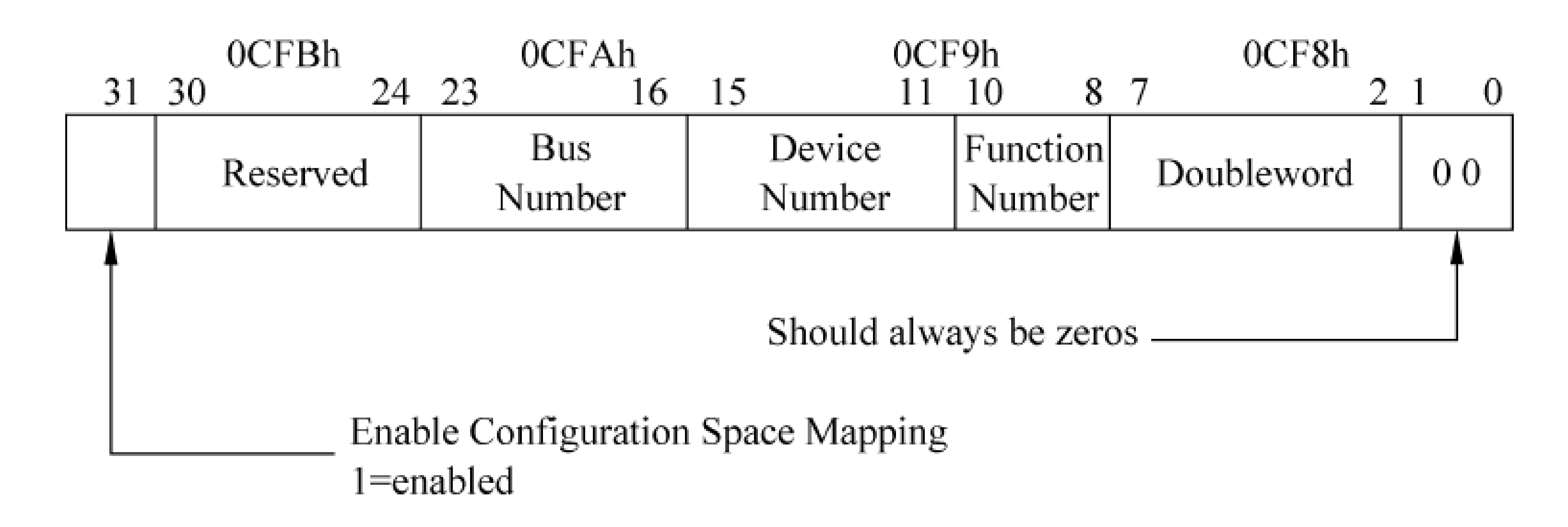


图 2.3　地址 0CF8h 上的 PCI 配置地址端口

表 2.2　增强配置机制存储器映射的 I/O 地址格式

存储器地址的 bit 位	说　明
A[63:28]	256MB 对齐的基址高位比特，由固件 BIOS 根据可用内存空间分配
A[27:20]	目标总线号（256 个中的一个）
A[19:15]	目标设备号（32 个中的一个）
A[14:12]	目标功能号（8 个中的一个）
A[11:2]	目标双字号（1024 个双字中的一个）
A[1:0]	定义所选双字的起始字节的位置

（2）PCI 总线枚举

PCI 配置需要知道配置寄存器的总线号和设备号。在系统加电时，固件仅仅知道总线 0 的存在（即驻留在主机/PCI 桥下游一侧的总线），但对于总线 0 上存在着什么样的设备毫不知晓，如图 2.4 所示[49]。固件需要对总线 0 进行扫描，发现不同总线及其上的设备和功能，找出 PCI 总线拓扑结构，这个过程称为枚举（Enumeration）。

PCI 总线枚举过程中，采用深度优先搜索算法，通过读取各台设备中功能 0 的厂商 ID 的返回值判断设备是否存在（0ffffh 代表无效设备），而通过读取头类型寄存器中的数值判断设备是单功能还是多功能设备，是端点设备还是桥设备，从而完成对总线 0 上挂载的其他主总线、二级总线和设备进行标记，建立设备列表。图 2.5 所示为总线枚举后的结果[49]。

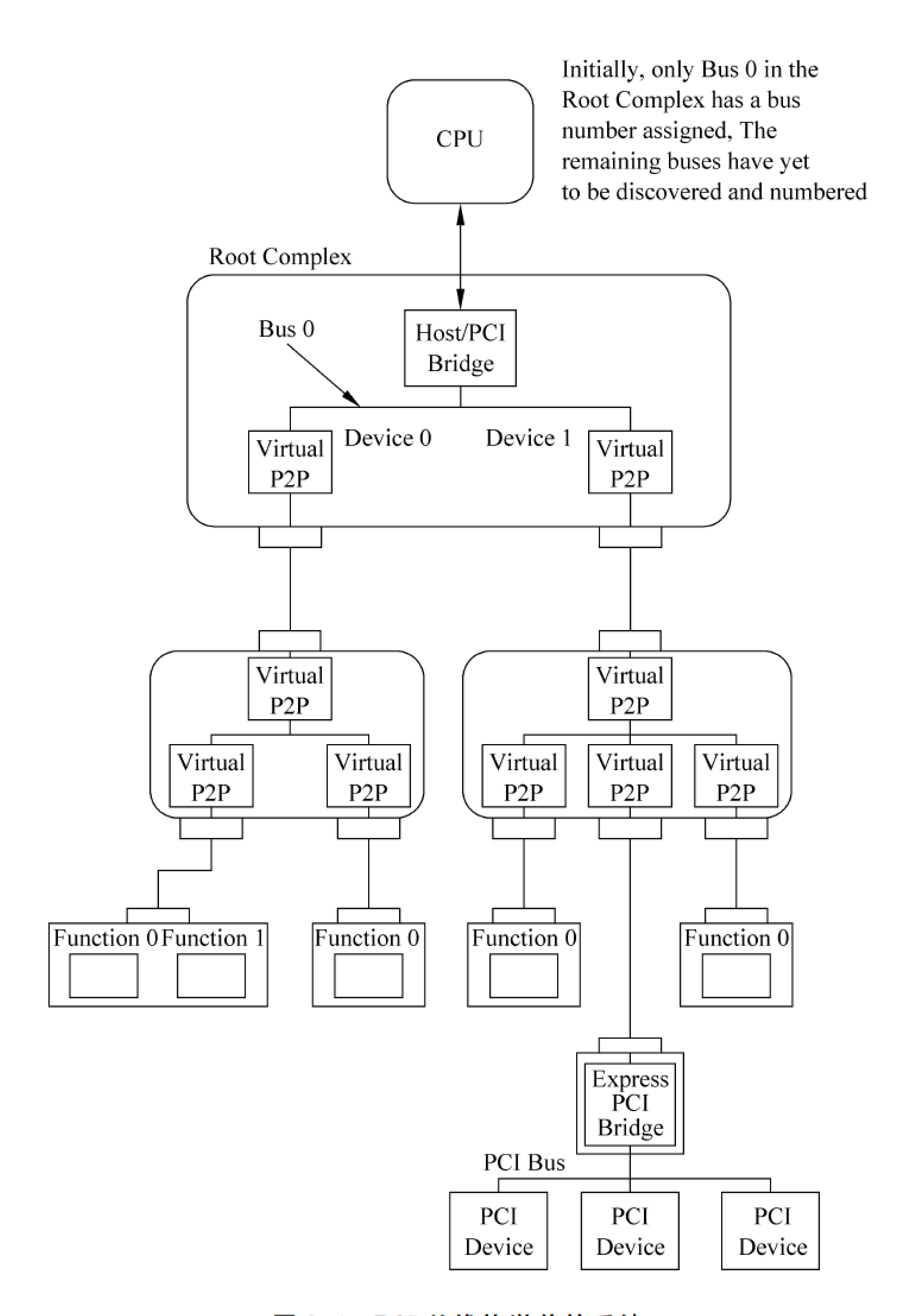


图 2.4　PCI 总线枚举前的系统

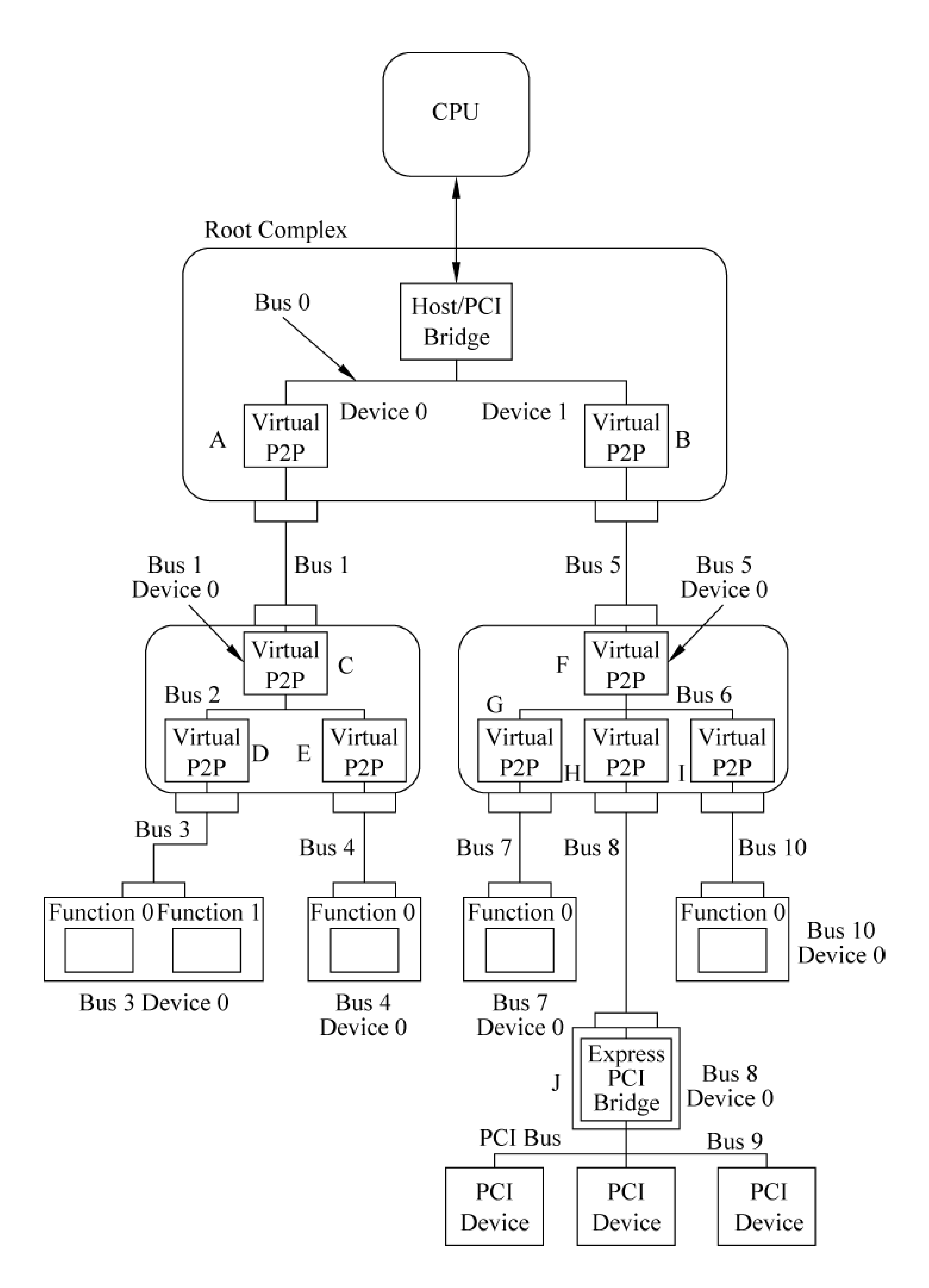


图 2.5　PCI 总线枚举后的系统

2.3.3 固件相关管理接口规范

在计算机传统固件 BIOS 发展过程中，其中涉及一些规范是需要硬件厂商、固件 BIOS 厂商以及操作系统厂商通力合作，在各自的产品中都能够遵循同一个规范，使得硬件、固件和操作系统能够协同一体工作完成某些功能要求。其中典型的规范包括：高级电源管理 APM 规范、即插即用 PnP 规范、系统管理 SMBIOS 规范、高级配置与电源管理接口 ACPI 规范。其中的一些规范目前已经被更新更好的规范所代替，只在少数情况下需要保留对旧的规范的兼容支持，如 APM 规范目前基本被 ACPI 取代，而 PnP 的规范则部分被包含在 ACPI 中。作为硬件和操作系统之间的联系纽带，固件产品的开发中重要的一部分就是遵循和实现这些规范。而其中 PnP 规范和 ACPI 规范，由于其作为固件同操作系统交互的接口作用，更是同固件安全技术紧密相关。

1. ACPI 规范[50]

ACPI(Advanced Configuration and Power Interface)1.0 在 1996 年年底发布，用来取代原有的 APM(Advanced Power Management)规范，但是直到 2000 年发布了 ACPI 2.0，这个规范才得到广泛的部署。

ACPI 规范是一个复杂的标准，其实现需要硬件、固件、操作系统的各层协作。计算机固件产品必须能够产生各种 ACPI Table，并且向操作系统报告 ACPI Table 的内存占用情况。而各种外围设备、控制器(如软盘、硬盘、光驱、网卡等)则需要随时将自身的电源使用及开关状态记录在 ACPI 寄存器中，主板中的传感器也会随时将各芯片的最新电压、温度等参数通过 ACPI 表格和 SMBus 总线回传给操作系统。固件还要为系统在各种不同状态下的启动设计和选择正确的路径。图 2.6 显示了 ACPI 体系结构。

ACPI 针对 CPU、各种不同外围设备等分别定义了多种电源状态，规定了各种电源状态的切换模式，如图 2.7 所示。如 CPU 的电源使用模式，从轻微省电的 S1、S2，

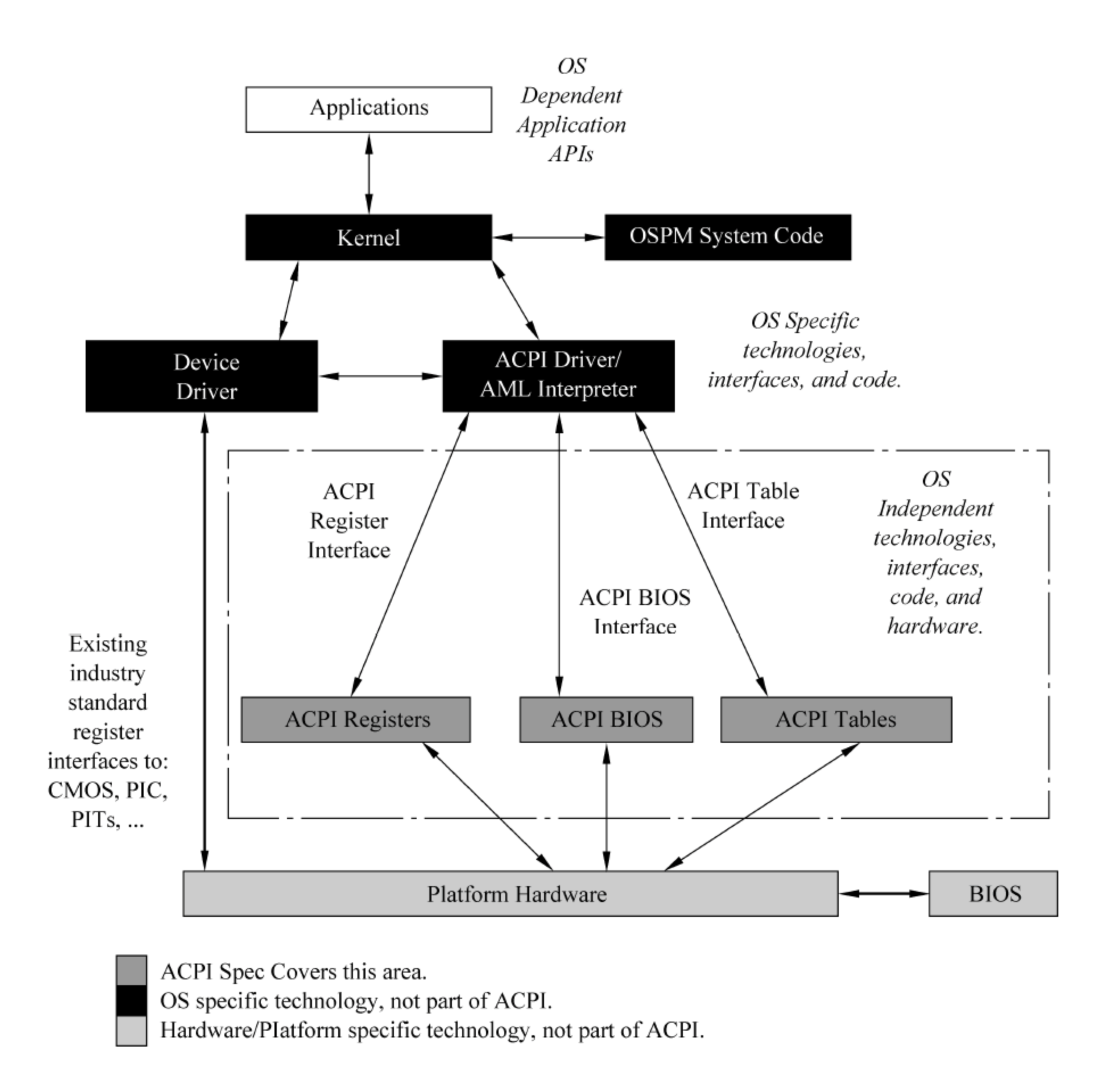


图 2.6　ACPI 体系结构

到了 S3 已经是 Suspend to RAM 模式，S4 则是 Suspend to DISK 模式，S5 就是真正的关闭电源。

对于固件开发者来讲，重要的是建立各种 ACPI 表格，根据平台特性收集并填写周边设备的 ACPI 信息；了解主板布线结构，设计必要的 ASL(ACPI Source Control Language)语言代码等。ASL 代码可由微软提供的 ASL 编译器编译生成 AML (ACPI Machine Control Language)目标代码，存储在固件产品运行时构造的 ACPI Table 中。

固件对ACPI的支持是一项琐碎、高技术、高难度的工程，通常固件厂商都要培养自己专业的ACPI工程师，来针对各种不同硬件平台完成固件中的ACPI支持部分。

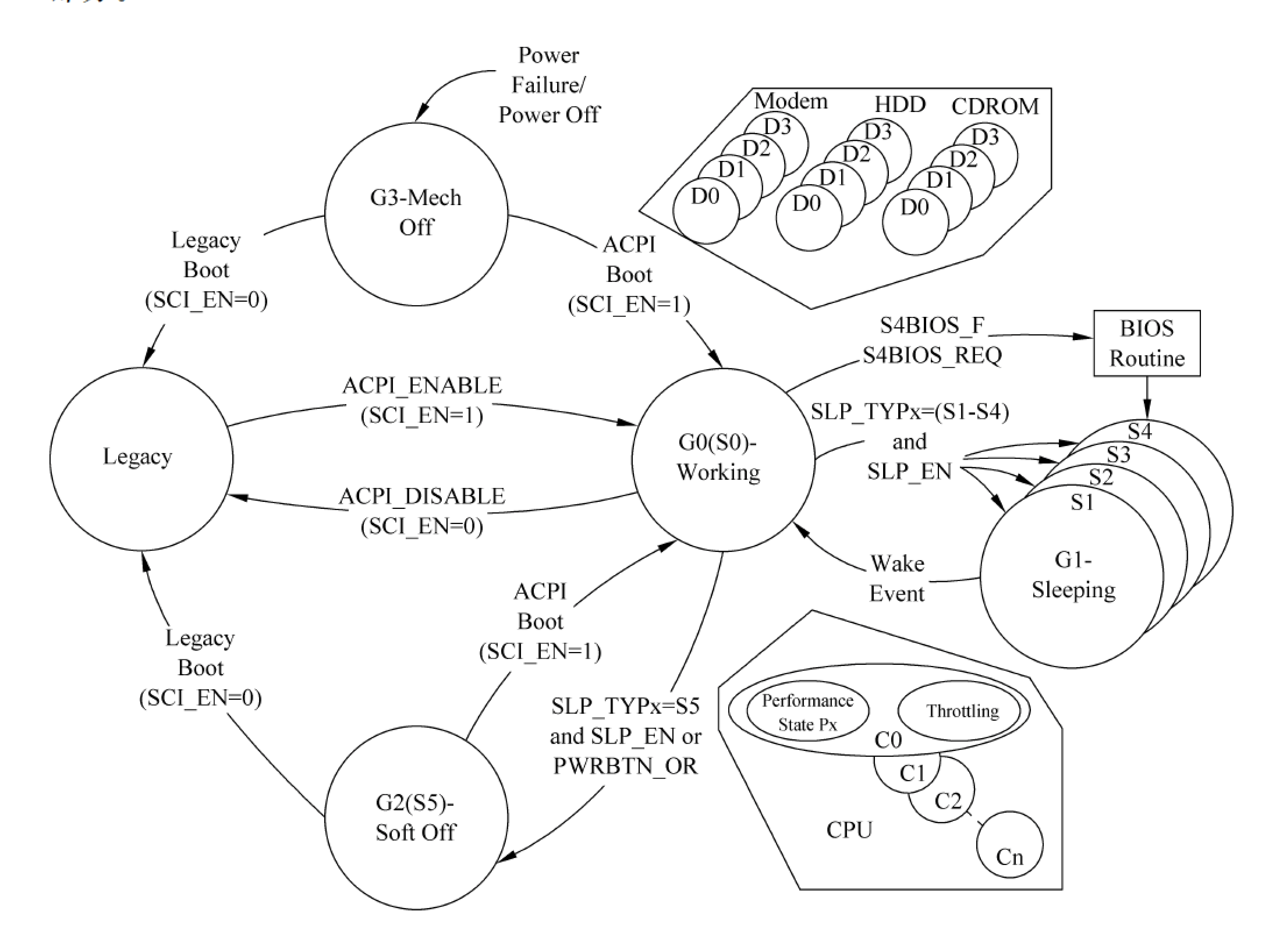


图 2.7 ACPI 电源状态模式图

2. SMBIOS[51]

SMBIOS(System Management BIOS)规范规定了在x86系统架构下，主板和系统制造商如何通过固件BIOS扩展接口，以标准格式提供产品的管理信息。固件开发者必须遵照规范，收集足够多的平台信息，提供固件扩展接口，以向用户或上层操作系统提供这些信息。SMBIOS规范最新的正式版本是2006年9月发布的2.5版本。

SMBIOS 信息的核心是一个 SMBIOS 结构表。该表存在于 0F0000h 到 0FFFFFh 内存空间中，以对齐于 16 字节边界的“_SM_”标识该表。表内容由表头和表结构信息组成。结构信息的内容包含最多 128 类的信息，由 Type 0 到 Type 127。

固件运行时要尽可能多地收集和填写规范中的这些信息，并且实现 SMBIOS 结构信息访问存取的接口函数。这些接口函数通常用于操作系统向固件索取这些系统信息。

针对固件的安全技术，通常也需要获取 SMBIOS 中的这些关键的系统信息。

2.3.4 固件内存管理与资源分配

传统固件 BIOS 对内存、I/O 端口、IRQ(Interrupt ReQuest)、DMA 通道、CMOS RAM、中断向量表(Interrupt Vector Table)、BIOS 数据区(BIOS Data Area，BDA)、BIOS 扩展数据区(Extended BIOS Data Area，EBDA)等资源的管理分配，由于沿用的习惯和兼容性的要求，形成了固定的规划[1,44,45]。新一代 EFI 固件的设计在内存使用上没有沿用这些固定规划，但是为了兼容传统的非 EFI 规范的硬件插卡的使用和操作系统的引导，必须由 EFI 固件厂商另外提供一个称为兼容性支持模块(Compatibility Support Module，CSM)[52]的产品，来实现对传统固件 BIOS 这些固定规划的兼容支持。

了解这些内存规划和映射区域结构及其内容分配，通过使用不层次的内存读写技术，就可以获取系统软硬件的关键参数资料，甚至复制内存映射中的固件关键代码和数据区域，这对实现固件的安全尤其重要，而这也成为恶意者攻击固件的前奏技术。

2.3.5 UEFI 固件框架和规范

可扩展固件接口(Extensible Firmware Interface，EFI)是 Intel 公司为全新类型的计算机固件的体系结构、接口和服务提出的建议标准。其主要目的是为了提供一

组在操作系统加载之前(启动前)在所有平台上一致的、正确指定的启动服务,被看做是有近20多年历史的计算机固件BIOS的继任者。

UEFI是由EFI 1.10为基础发展起来的,它的所有者已不再是Intel公司,而是一个称作Unified EFI Forum的国际组织,贡献者有American Megatrends Inc.、Apple Computer,Inc.、Dell、Hewlett Packard、IBM、Insyde、Intel、Lenovo、Microsoft、Phoenix Technologies等几个大公司。UEFI的框架和规范实现代码是完全开源的,但是具体的硬件平台代码还是需要主板厂商跟固件厂商一起做一些定制。与传统固件BIOS相比,最大的几个区别在于:

(1) 编码99%都是由C语言完成。

(2) 一改之前的中断、硬件端口操作的方法,而采用了Driver/protocol的新方式。

(3) 不再支持x86实模式,而直接采用Flat mode(也就是不能用DOS了,现在有些EFI或UEFI能用是因为做了兼容)。

(4) 输出也不再是单纯的二进制代码,改为Removable Binary Drivers。

(5) 操作系统引导不再是调用Int19,而是直接利用protocol/device Path。

(6) 对于第三方的开发,传统固件BIOS基本上做不到,除非参与BIOS的设计,但是还要受到ROM芯片大小的限制,而UEFI就便利多了。

(7) 弥补传统固件BIOS对新硬件支持不足的缺陷。

UEFI结构可以划为两部分:UEFI的实体(UEFI Image)和平台初始化框架。

根据UEFI规范定义,UEFI Image包含三种:UEFI Applications,OS Loaders和UEFI Drivers。UEFI Applications是硬件初始化完成后,操作系统启动之前的核心应用,如启动管理、BIOS设置、UEFI Shell、诊断程式、调度和供应程式、调试应用等。OS Loaders是特殊的UEFI Application,主要功能是启动操作系统并退出和关闭UEFI应用。UEFI Drivers是提供设备间接口协议,每个设备独立运行提供设备版本号和相应的参数以及设备间关联,不再需要基于操作系统的支持。

UEFI框架主要包含两部分,一个是EFI与初始化PEI(Pre-EFI Initialization),另一部分是驱动执行环境DXE (Driver Execution Environment)。PEI主要是用来

检测启动模式、加载主存储器初始化模块、检测和加载驱动执行环境核心。DXE 是设备初始化的主要环节，它提供了设备驱动和协议接口环境界面。

目前，UEFI 主要包括以下四个规范：

- UEFI 规范（UEFI Specification）；
- UEFI 平台初始化规范（UEFI Platform Initialization Specification）；
- UEFI Shell 程序规范（UEFI Shell Specification）；
- 自测试规范（Self-test Specification）。

2.4　本章小结

本章首先考查了计算机固件产品和技术的发展历程，按照产品技术特色和功能将计算机固件发展划分为传统固件 BIOS 和新一代 EFI 固件，其中传统固件 BIOS 又被划分为第一代传统固件 BIOS 和第二代传统固件 BIOS。2003—2007 年间为传统固件 BIOS 产品和新一代 EFI 固件产品共存的发展阶段，而从 2008 年开始，EFI 固件产品加速取代传统固件 BIOS 产品。而从长远来看，新一代 EFI 固件取代传统固件 BIOS 已经是产业不可逆转的发展趋势。

近几年来，尽管存在着为数不少的传统固件 BIOS 开源开发项目，但受处理器、芯片组以及其他硬件设备技术资料非公开、半垄断或垄断限制，开源固件 BIOS 产品通常具有支持的硬件平台少、功能简单、缺乏对诸多工业规范如 ACPI、SMBIOS 的支持、技术落后等缺陷。EFI/UEFI 规范作为新一代固件事实上的标准，技术较传统固件 BIOS 更为先进，其公开的框架代码既实现了对 EFI/UEFI 规范的支持，又实现了对诸多工业规范的支持，因此是开发新一代安全固件首选技术路线。

本章所介绍的固件开发关键基础技术和规范，既是后续章节对传统固件 BIOS 存在的安全漏洞、安全威胁进行研究分析，增强固件安全的必备基础，也是安全固件开发实践的必备基础。



第 3 章 固件安全技术研究开发实例

传统固件 BIOS(Legacy BIOS)产品的核心技术,历来只在计算机制造产业链的有限环节有限开放,BIOS 厂商掌握这些技术,同时向合作的主板厂商、整机厂商按级别提供部分所需的技术资料,并要求这些合作厂商对这些资料进行保密。这种封闭性使得传统固件 BIOS 发展的 30 多年来,改变甚少,技术进步缓慢。其他应用厂商要想在固件层次做出一些应用产品也相当困难。一些高明的黑客,或者一些第三方应用厂商,采用逆向工程技术破解和掌握这些有限种类的传统固件产品的相关技术,以求能够在固件层次对计算机发动攻击,或者利用固件层次增强和保护计算机安全。作者正是基于这一背景,在 2002 年前后开始采用逆向工程技术,结合有限的开放技术资料,以求在固件层次研究开发一些安全工具程序。本章利用作者研究开发的两个固件安全程序作为实例,分别讲述在 Award BIOS 和 Phoenix BIOS 这两种产品中,开发固件安全程序的技术原理、方法和过程。

3.1　Legacy BIOS 固件安全增强技术

我国台湾地区厂商自主开发的 Award BIOS 一度是在个人计算机上应用最广泛的一种公用固件产品，后来在 1998 年被美国 Phoenix 公司收购，作为 Phoenix 公司主流的个人计算机固件产品。本节讲解针对 Award BIOS 固件产品自主开发的一种增强个人计算机安全保护的安全工具：固件刷卡开机工具。这种增强工具包含一个固件模块程序，该固件模块程序被嵌入到计算机主板上的 BIOS flash 芯片中。当计算机上电启动后，在 BIOS 固件运行阶段，嵌入在 flash 芯片中的固件模块程序会被执行，要求用户使用指定的 IC 卡刷卡开机。如果刷卡正确则继续开机过程，进入操作系统运行，否则停止开机过程。通过这个实例，让读者了解和掌握 Award BIOS 固件应用程序模块的结构、编写方法、嵌入方法，并因此能够编写更多的固件安全程序。

3.1.1　固件刷卡开机原理与流程

固件层次的刷卡开机，是指在计算机主板上的 BIOS flash 芯片中嵌入一个刷卡开机模块程序。计算机加电开机后，在固件 BIOS 运行阶段，截获计算机运行控制权，等待用户刷卡验证。只有验证正确，才继续 BIOS 运行阶段的加载操作系统的过程（也就是传统的 INT 19 过程），否则停止运行。通过刷卡开机验证，增强计算机安全特性。特别是该安全模块嵌入在 BIOS flash 芯片中，难以破解和绕过，相比操作系统层次的开机验证，提供了更好的安全性。

加入刷卡开机验证后，计算机固件 BIOS 运行流程为：

① 计算机开机上电；

② BIOS 自检（BIOS POST 阶段，Power on Self Test）；

③ BIOS 初始化，中断可用；

④ 执行刷卡开机安全增强模块；

⑤ 刷卡验证,若验证成功,转向步骤⑥,否则停止开机过程;

⑥ 释放运行控制权给 BIOS,继续 BIOS 运行其他过程。

在上面的流程中,重要的一点是一定要使步骤④执行刷卡开机安全增强模块开始运行前,固件 BIOS 运行过程已经完成对计算机的中断初始化过程。因为该安全增强模块中需要调用运行某些 BIOS 中断程序。这个问题可以通过指定安全增强模块的类型来解决,后面会讲述到。

刷卡开机安全增强模块的执行流程为:

① 设置显示模式;

② 显示刷卡验证提示;

③ 初始化串行通信端口 1,准备读入卡内容;

④ 验证卡内容,若正确,结束模块执行,返回 BIOS 运行;若 3 次验证都失败,停机。

3.1.2 编写固件安全增强模块程序

下面程序的编写,只关注于编写正确的可执行的 Award BIOS 固件模块的结构和刷卡验证的功能,删除了原来开发产品中的其他功能程序部分。

```
;awdexample.asm
.model tiny                          ;汇编后得到.com 程序,而不是.exe 程序
.386
;定义宏 disp_string,用于在屏幕上显示一行提示信息
disp_string macro str_text,str_len,row,col,attr
    push ax
    push bx
    push cx
    push dx
    push bp
    mov ah,13h
    push cs
    pop es
    lea bp,str_text
```

```
      mov cx,str_len
      mov bh,0
      mov dh,row
      mov dl,col
      mov al,1
      mov bl,attr
      int 10h
      pop bp
      pop dx
      pop cx
      pop bx
      pop ax
      endm
text_color        equ 0ah          ;定义显示的字符的颜色

;采用 DOS 下 COM 程序的格式
code      segment
          assume cs: code,ds: code,es: nothing,ss: nothing
          org      0h
          db 55h,0aah              ;ISA 模块标签
          db 40h                   ;模块长度,以 512 字节为单位
                                   ;为避免复杂计算,此处干脆定义为 32KB,不影响程序运行
start:
    db 66h,60h,1eh                 ;保存寄存器
    push cs
    pop ds
    push cs
    pop es
;set video mode
    mov ah,00
    mov al,12h
    int 10h
    mov ah,0bh
    mov bh,0
    mov bl,1
    int 10h

    call get_passwd                ;调用读卡验证子程序
    db 1fh,66h,61h                 ;恢复寄存器
    retf                           ;返回 BIOS 继续执行,一定要用长返回
; 初始化串行通信端口 comm1
```

```
i8250 proc near
        push ax
        push dx
        mov dx,3fbh
        mov al,80h
        out dx,al                ;访问除数寄存器,波特率 9600
        mov dx,3f9h
        mov al,0
        out dx,al                ;除数寄存器高字节
        mov dx,3f8h
        mov al,0ch
        out dx,al                ;除数寄存器低字节
        mov dx,3fbh
        mov al,03h
        out dx,al                ;通信控制器 1 位停止位,无校验,8 位数据位
        mov dx,3f9h
        mov al,0
        out dx,al                ;中断允许控制器,查询方式
        mov dx,3fch
        mov al,0
        out dx,al                ;modem 控制器,正常工作方式
        pop dx
        pop ax
        ret
i8250 endp

;读卡验证子程序
get_passwd proc near
   mov bl,0                      ;row for writing 字符串显示起始行
   mov bh,3                      ;最大验证次数

;display waiting message
   disp_string passwd_tip,passwd_tip_len,bl,0,text_color
   inc bl

;初始化 comm1
     call i8250
;IC 卡内验证号为 10 位,从 comm1 读 10 个字节
again:
   push cs                       ;set destination string buffer
```

```
    pop es
    lea di,comm_buffer
    cld
    mov cx,11
check:
        mov dx,3fdh             ;if error
        in al,dx
        test al,1eh
        jnz read_fail
        test al,01h             ;receive data ready
        jz check
        mov dx,3f8h             ;receive data
        in al,dx
        and al,7fh
        stosb
        loop check
        jmp check_passwd

;display string for read fail
read_fail:
    disp_string comm_error,comm_error_len,bl,0,text_color
    inc bl
    jmp again

;比较刷卡读入的内容和指定的开机卡内容
check_passwd:
    inc bl
    lea di,comm_buffer
    lea si,passwd
    push cs
    pop ds
    cld
    mov cx,10
    repe cmpsb                  ;预先指定卡内容与读入卡内容比较
    jz return                   ;验证成功,正常返回主程序
    dec bh
    jz fail                     ;3 次都错误,验证失败,停机
    ;失败不足 3 次,显示提示信息,继续下一次刷卡验证
```

```
    disp_string passwd_try,passwd_try_len,bl,0,text_color ;passwd mismatch
    inc bl
    jmp again

;显示验证失败信息,停机
fail:
    disp_string passwd_fail,passwd_fail_len,bl,0,text_color
halt:
    hlt
    jmp halt

; 正常返回主程序
return:
    ret
get_passwd endp

;data for get_passwd
passwd_tip          db 'waiting for password......'
passwd_tip_len          dw $-passwd_tip
passwd_try          db 'error! try again......'
passwd_try_len          dw $-passwd_try
passwd_fail         db 'sorry,you have no right to use this computer!'
passwd_fail_len         dw $-passwd_fail
comm_buffer         db 11 dup(0)
passwd              db '18006DABD4'
comm_error          db 'fail when read from comm,try again'
comm_error_len          dw $-comm_error

code     ends
end      start
```

使用 MASM6.11 汇编工具把上述代码文件 awdexample.asm 汇编得到 awdexample.com 程序文件,该.com 文件即成为可以嵌入到 Award BIOS 固件中的合法程序。

总结起来,编写第三方 Award BIOS 固件应用程序要点包括:

(1) 一定要是一个.com 程序文件结构,程序只能包含一个段,程序长度不能超过 64KB。

(2) 务必要加上 ISA 模块程序标签。因为 Award BIOS 将 ISA 模块同主板上 ISA 卡中的 ROM 程序等同处理，ISA 模块程序执行时，Award BIOS 已经处理好了绝大部分的初始化工作，包括大部分的 BIOS 中断初始化，此时方可放心地调用大部分 BIOS 中断程序。否则太早执行，会导致可利用资源少，程序编写可用的中断和其他资源受到较大的限制，导致程序运行错误。

(3) 模块执行结束必须使用 RETF 远程返回，而不能使用 RET 或其他正常.com 程序那样的 DOS 中断返回，因为 award BIOS 对 ISA 模块调用采用的是一种远调用(FAR CALL)。

(4) 由于操作系统尚未加载启动，许多在操作系统环境下编程使用的语句和资源尚不能使用，特别是操作系统级的中断程序调用，这一点千万注意！否则程序汇编不出错，但嵌入到 BIOS 芯片中运行就会发生不可预知的错误，导致程序运行跑飞。

3.1.3　在 BIOS flash 芯片中嵌入安全增强程序

得到的 awdexample.com 文件，尚不能直接嵌入到 BIOS flash 芯片中，这是因为 Award BIOS flash 芯片中的内容是有固定的组织结构的。读出 Award BIOS flash 芯片中的内容存成文件(称为 BIOS flash 映像文件)，可以发现它的模块程序都是由模块头部结构和模块的数据/代码两部分构成，并且模块数据/代码部分往往是经过压缩存储的。关于 Award BIOS 典型的模块头部结构在本书 5.5.1 节中介绍。

使用 Award 公司提供的工具程序 CBROM 可以把 awdexample.com 程序转换成 Award BIOS 模块，并可以将该模块嵌入到 BIOS flash 映像文件中的空白空间处(不用担心映像文件中没有空白空间或空白空间不足，在作者十多年固件研究过程中，这种情况还从来没有遇到过，因为主板厂商总会慷慨地提供多余的空白 flash 芯片空间以防万一)。

在使用 cbrom 工具操作之前，需要先得到 BIOS flash 芯片映像文件，即读出 BIOS flash 芯片内容并存储成数据文件。有很多可用的软件工具可以帮助完成这项工作，如 awdflash，uniflash，winflash 等。并且这些工具理所当然地可以完成相反的

工作，即把修改后的映像文件再写回到主板上的 BIOS flash 芯片中。

因此，要想把上面完成的固件安全增强模块程序嵌入到计算机主板上 BIOS flash 芯片中，可以采用下面描述的过程来完成。

1. 读取 BIOS flash 芯片内容，得到 BIOS flash 映像文件

运行命令：

```
awdflash mybios.bin mybios.bin /pn/sy
```

如果想要看到命令执行过程中的屏幕提示信息，可以删除参数/pn/sy 来运行这条命令。这条命令执行的结果，是把主板上 BIOS flash 芯片中的内容读出来并存储到当前目录下的 mybios.bin 文件中。

2. 把安全增强模块程序嵌入到 BIOS flash 映像文件中

运行命令：

```
cbrom mybios.bin /isa awdexample.com
```

这条命令执行的结果，是把 awdexample.com 转化成符合 awawd BIOS 模块结构的 ISA 代码模块(加上模块头部结构，并且头部结构中模块类型标识值为 0x40A40000，即 ISA 模块)，并将该模块嵌入到映像文件 mybios.bin 中的空白区部分。

当然，如果是初次试验，建议先备份映像文件 mybios.bin，然后再作上面的嵌入操作，以免发生意外时有可以用来恢复 BIOS flash 芯片的正确的映像文件。

3. 用嵌入了安全增强模块程序的映像文件刷新 BIOS flash 芯片

运行命令：

```
awdflash mybios.bin /py/sn/E
```

这条命令执行的结果，就是用已经嵌入了安全增强程序 awdexample.com 模块的映像文件 mybios.bin 刷新主板上的 BIOS flash 芯片，使得主板上 BIOS flash 芯片中包

含这个安全增强程序。

成功完成上面3个步骤后，现在可以重新启动你的计算机，享受刷卡开机带来的快乐吧。

上面这个刷卡开机的固件安全增强程序工具开发的描述，有助于此基础上完成更多复杂的基于Award BIOS固件产品的第三方安全应用。

3.2　Legacy BIOS固件安全代理技术

代理技术是指能自动后台运行某些应用的工具。固件安全代理，是指在固件中嵌入的安全代理程序，它能够在计算机固件运行后，自动被加载运行，完成事先指定的安全功能。这种固件代理技术可用于攻防两个方面，用于防的方面，可以在固件层次完成操作系统运行前后的安全代理服务功能，如远程验证、服务下载等；用于攻的方面，可以用于在固件中隐藏恶意代码（如木马或rootkit程序），并且可分别于固件运行阶段和操作系统运行阶段窃取系统运行控制权。

Phoenix BIOS固件，特别是Phoenix 4.0固件，是笔记本电脑上最主流的传统BIOS固件。本节阐述在Phoenix 4.0 realease 6.0版本的固件产品中，开发一个固件安全代理程序的原理、方法和编程技术。由于针对的固件产品的不同，使得这个固件安全代理开发技术和3.1节的固件安全增强程序开发有较大的不同。

该固件安全代理嵌入在计算机主板上的BIOS flash芯片中。在BIOS运行阶段，该固件安全代理会被执行，并且将自身的一部分功能服务程序释放到硬盘中的操作系统中，并使得操作系统加载运行后，能自动地在后台运行由固件释放过来的安全代理功能服务程序。由于这种安全代理隐身于固件flash芯片中，因此即使执行删除重装操作系统、更换硬盘等操作，也不会影响这种固件安全代理的释放和执行。

3.2.1 固件安全代理技术原理与流程

固件安全代理程序采用了两层结构，外层程序称为 shell 程序，是一个符合 Phoenix 4.0 BIOS 固件格式的固件模块程序，在 BIOS 运行阶段被执行。内层程序称为 stone 程序，是一个 Windows 操作系统可执行程序，在 Windows 操作系统启动后被自动加载执行。在 BIOS 运行阶段，shell 程序被加载执行后，其主要工作包括：一是在硬盘的启动分区查找操作系统安装目录，把自身包含的 stone 程序模块释放并写入到硬盘中的操作系统安装目录中；二是修改操作系统配置文件 system.ini 文件，使得所释放的 stone 程序在操作系统加载启动后能够在后台自动执行。而固件安全代理的代理服务功能则由 stone 程序在 Windows 运行环境下完成，目前该代理服务功能主要是同远程的指定服务器连接验证，完成服务器提供的上传下载文档的指示。

加入固件安全代理程序后，计算机固件 BIOS 运行流程为：

① 计算机开机上电；

② BIOS 自检(BIOS POST 阶段，Power on Self Test)；

③ BIOS 初始化，中断可用；

④ 按照固件模块链接表顺序，执行固件安全代理程序模块；

⑤ 固件安全代理模块释放，运行控制权给 BIOS，继续 BIOS 运行其他过程。

固件安全代理程序模块的执行流程为：

① shell 程序运行；

② 识别硬盘操作系统分区类型；

③ 若操作系统分区类型被支持，将所包含的 stone 程序写入到硬盘中系统目录中；若操作系统分区类型不被支持，转向步骤⑤；

④ 在硬盘系统目录中查找系统文件 system.ini，添加自启动 stone 程序配置项内容；

⑤ 返回 BIOS 继续执行。

3.2.2　编写安全代理 shell 模块程序

下面的缩略程序，主要是为说明在 Phoenix 4.0 BIOS 固件中，如何编写第三方可执行的固件应用程序模块，以及说明安全代理中 shell 程序和 stone 程序之间的依存关系。而关于磁盘文件系统在固件中的读写处理，把 stone 程序写入到磁盘上操作系统目录中等磁盘文件处理操作代码，由于与本书研究重点关系不大，因此，只保留简要的磁盘扇区读写处理部分代码，以示固件中磁盘读写处理方法。

```
;phshell.asm
.model tiny
.386
code      segment
          assume cs: code,ds: code,es: nothing,ss: nothing
          org      0h
main:
   db '$INI&'                        ;标志其后的字节为开始执行的代码
   db 66h,60h,1eh                    ;保存寄存器
   ...                               ;省略 shell 程序显示和菜单初始化代码
   push cs
   pop ds
   push cs
   pop es
   sti                               ;开中断
   mov ax,0201h
   mov bx,offset diskbuffer
   mov cx,1
   mov dx,80h
   int 13h                           ;读磁盘 master boot 扇区

   mov si,offset diskbuffer
   mov dl,byte ptr [si+1c2h]         ;读取分区类型
   cmp dl,0bh                        ;分区类型为 fat32
   je fat_32
   cmp dl,0ch                        ;分区类型为 fat32(LBA)
   je fat_32
```

```
    jmp return                          ;其他分区类型,暂时忽略,不做操作

fat_32:
    mov si,offset diskbuffer            ;判断是否活动分区 (active partion)
    mov dx,word ptr [si+1beh]           ; (00h=Inactive,80h=Active)
    cmp dl,80h                          ;是活动分区
    je stone
    jmp return                          ;非活动分区,忽略,不做操作

;下面省略释放 stone 程序模块到活动分区系统目录的过程
stone:
    ...

return:
    db 1fh,66h,61h                      ;恢复寄存器
    retf                                ;把系统运行控制权交回给 BIOS
;省略其他子程序和数据定义
...

stone_flag  db "STONE_BEGIN"            ;stone 程序存放起始标记
            db 5000h dup(042h)          ;stone 程序模块存放的预留空间
            diskbuffer db 512 dup(0)    ;磁盘扇区数据缓冲区

code    ends
end     main
```

从上述 phshell. asm 程序可以发现,编写 Phoenix 4.0 BIOS 固件产品的可执行程序模块同编写 Award BIOS 固件产品的可执行 ISA 程序模块相比较,程序结构既有明显的不同,又有相似之处。其中最主要的不同在于两者程序模块开始处头几个字节的含义不同。

在安全代理 shell 程序中,为安全代理 stone 程序模块预留了数据缓冲区。当编写好 stone 程序后,就可以把 stone 二进制程序嵌入到 shell 模块中这部分预留数据缓冲区中。这样,当 shell 程序在 BIOS 运行阶段被执行时,就可以读取这部分 stone 二进制程序,把它写入到磁盘文件中。在固件执行阶段,对于磁盘数据的读写操作,可以采用 BIOS 中断 INT 13h 来完成。

至于安全代理的 stone 程序,因为其完全是一个 Windows 下的可执行程序,在这

里不再讨论其编写技术。但是需要提醒的是，在 BIOS flash 固件芯片中寸土寸金的地方，是没有足够的空间容纳庞大的 Windows 程序的。因此 stone 程序的编写，必须要考虑采用减少其二进制程序长度的技术。

3.2.3　在 BIOS flash 芯片中嵌入安全代理程序

准备好安全代理程序模块后（包括 shell 程序和 stone 程序），下面要做的工作是把这个安全代理程序模块嵌入到计算机主板上的 BIOS flash 芯片中。这个过程与 3.1.3 节在 Award BIOS flash 芯片中嵌入安全增强程序也很相似，只是所采用的工具有所不同，两者模块结构也不同。

(1) 把 stone 二进制程序插入到 shell 二进制程序模块中的预留空间中，得到完整的安全代理程序模块。

(2) 把完整的安全代理程序模块处理成为合法的 Phoenix 4.0 BIOS 固件可执行模块，即加上模块头部结构，并对程序模块进行压缩处理（也可以不压缩）。关于 Phoenix 4.0 BIOS 固件模块头部结构，本书 5.5.2 节再作更详细的介绍。这个步骤的操作可以自己编写 Windows 小工具程序来实现，也可以使用 Phoenix 公司提供的 prepare 工具来实现。假设完整的安全代理程序模块文件名为 module.rom，使用 prepare 工具实现，只需要运行下面的命令行：

```
prepare module.scr
```

其中 module.scr 是一个文本类型的配置文件，其内容可以编辑如下：

```
COMPRESS   LZINT             #使用 LZINT 压缩算法压缩模块内容
#no compress
#SETUP module.ROM-X          #无需压缩模块内容，删除这个 setup 之前的注释符
#compress
#SETUP module.ROM            #要压缩模块内容，删除这个 setup 之前的注释符
```

prepare 命令运行后生成的新的模块的名称为 module.mod。module.mod 模块就是一个可以嵌入到 Phoenix 4.0 BIOS 固件芯片中的合法的可执行模块。模块类型

为 setup，类型标识符值为 E。

(3) 获取计算机主板上的 Phoenix 4.0 BIOS 固件映像文件。这可以通过使用 uniflash 软件工具来得到。Uniflash 工具是一个开源的读写主板 BIOS 固件芯片的工具。

(4) 把完整的合法的安全代理程序模块 module.mod 嵌入到 BIOS 映像文件中。这一步操作没有现成的可用工具，无论是 Phoenix 公司还是其他第三方，都没有提供公开的工具来实现这个功能。作者通过逆向分析和反复试验，终于成功地完成这个步骤的操作，并且编写了专用的小工具程序 foister.exe 来自动完成这个步骤的操作。工具程序 foister.exe 的操作流程将在下面进一步阐述。

(5) 把嵌入了安全代理程序模块的 BIOS 映像文件刷新到主板上 BIOS flash 芯片中。这一步骤操作同样可使用开源软件工具 uniflash 来完成。不再赘述。

经过上面的五个步骤的操作，以后每次开机启动后，BIOS flash 中的安全代理程序都会被执行，并且把芯片中包含的 stone 代理模块程序释放到 Windows 系统中(实际中加入了重复释放检查)。Windows 操作系统启动后，会在后台自动执行 stone 代理程序，该代理程序与远程指定服务器进行验证和连接，完成服务器要求的文档上传和下载指示。并且只要不更换主板，即使重新安装操作系统、格式化硬盘、更换硬盘，这个固件安全代理程序也会依然存在。同样的道理，这样的一个固件安全代理程序如果用于攻击，很容易演化成为一个固件木马程序。

下面描述一下上述第(4)步中提到的工具程序 foister.exe 的功能流程。foister.exe 主要功能是在映像文件中查找空白区(全 0xFF 覆盖的区域)，找到后把安全代理程序模块 module.mod 的内容覆盖到这片空白区中，然后在映像文件中根据模块链表查找固件中原有的 setup 模块，找到后通过修改其前后的模块头部连接指针地址，以及安全代理程序模块头部指针地址，把安全代理程序模块插入到映像文件中的模块单链表存储结构中(见 5.5.2 一节 Phoenix BIOS 映像文件和模块结构)。

之所以选择把安全代理程序模块设置成 setup 类型的模块，是因为在 Phoenix 4.0 BIOS 固件中，setup 类型的模块被执行时，往往代表着 BIOS 固件的软硬件初始化工作都已经完成，固件应用程序可以放心地调用各种软硬件资源了。这对于安全代理

程序成功地运行是至关重要的。

3.3　本章小结

本章结合作者早期对固件安全的研究开发成果，分别阐述了在传统的 Award BIOS 和 Phoenix BIOS 这两种主流的个人计算机和笔记本电脑固件产品中，如何开发编写第三方安全应用程序，如何利用软件手段把第三方安全应用程序嵌入到计算机主板上的 BIOS flash 芯片中。使得读者能够基本上掌握这些固件应用程序开发的原理和技术手段，在此基础上，能够更好地理解后续章节的固件安全技术研究内容。

固件安全增强技术使得计算机安全在物理安全、操作系统安全、网络安全层次之外，又多了一层固件层次的安全保护。

固件安全代理技术则可以用于安全攻防的两个方面，特别地要关注其用于固件木马技术的开发。

同时，也希望本章的内容能引起更多人对固件安全和威胁更实在的感受和体验，让计算机固件安全在我国得到更大范围的重视。



第 4 章

固件BIOS安全漏洞及威胁研究

计算机固件 BIOS 的安全长期不被重视，其中一个重要的原因，是计算固件是固化在硬件芯片中的一种软件。因为固化在硬件芯片中得到硬件芯片的保护，因此要想对其代码和数据进行篡改就很困难。但用户的需求和技术的发展改变了这一切，使得其安全问题越来越多地袒露在公众面前，甚至于 CIH 病毒令到一般用户和专业人士都必须要重新审视固件安全威胁的严重后果。固件产品的安全漏洞同操作系统、应用软件系统中存在的安全漏洞相比较，在数量上少之又少，但绝对不可以忽略不计。针对固件的安全威胁技术，由于其深处底层，更有上层软件技术威胁无可比拟的摧毁性。

本章对固件安全漏洞进行了截至目前最为完整的分析，并分类对威胁固件安全的多种技术从原理和实现方法上进行了探讨。

4.1 固件 BIOS 安全漏洞和威胁概念的含义

CVE(Common Vulnerabilities and Exposures)是全球信息安全界对计算机安全漏洞和脆弱性统一命名的权威知识库。CVE 列表(CVE List)包含目前所有已知的

计算机安全漏洞和脆弱性及其标准命名。

在CVE文档中，安全漏洞和脆弱性两者的概念和外延是有区别的。

在CVE中，对“漏洞(Vulnerability)”的定义是：

定义4-1　信息安全领域的漏洞是指存在于软件中的错误，这种错误能够被黑客直接利用以获取对系统或网络的访问权限。

按照CVE的定义，主机或网络系统中存在的“错误”，只有该“错误”能够被攻击者直接利用并违反系统的安全政策时，才能够称为“漏洞”。例如，攻击者利用漏洞冒充合法用户执行命令，或访问只有合法用户才能访问的数据，或实施一次拒绝服务攻击(Denial of Service)。

CVE中对“脆弱性(Exposure)”的定义是：

定义4-2　信息安全领域的脆弱性是指存在于软件中的系统配置选项或配置错误，该配置选项或错误能够被黑客用于非法访问系统信息，或作为进一步获取访问或进入主机和网络系统的跳板。

按照CVE的定义，系统配置选项或错误的脆弱性不会直接危及系统的安全，但可能成为一次成功攻击的重要组成部分。例如，攻击者可利用系统脆弱性来收集攻击需要的相关信息、隐藏攻击活动的踪迹，或作为攻击的突破口等。

文献[28]给出了另一种外延更宽的安全漏洞定义：

定义4-3　安全漏洞(Vulnerability)是存在于计算机系统、网络系统的硬件，软件，或与系统相关的安全过程和控制方式中的设计或实现缺陷，可被有意或无意地利用从而危害组织或个人的财产或操作过程。

定义4-3所定义的漏洞(Vulnerability)的概念，同时涵盖了定义4-1和定义4-2的“漏洞(Vulnerability)”和“脆弱性(Exposure)”这两个不同的概念。实际上，在CVE列表(CVE List)中，也并没有对漏洞和脆弱性加以区分。

本书所指固件BIOS安全漏洞，采用定义4-3的安全漏洞概念。

定义4-4　威胁(Threat)是指有意或无意地利用系统存在的漏洞而破坏系统的机密性、完整性或可用性的潜在情形或事件[28]。

威胁通常由威胁代理(Threat Agent)来实现，恶意的黑客、犯罪组织、内部人员

(如系统管理者和开发者)、恐怖分子等都可能成为威胁代理。

固件 BIOS 安全威胁是指利用固件 BIOS 系统存在的漏洞,从而导致对计算机 BIOS 固件、操作系统等软件以及硬件设备和数据实施破坏的可能情形和事件。固件 BIOS 安全威胁可指来自于 BIOS 固件系统的威胁,也可指对 BIOS 固件系统构成的威胁。

4.2 固件 BIOS 安全漏洞和威胁的成因

固件 BIOS 安全漏洞和威胁是伴随着固件 BIOS 技术和产品的发展更新而逐步产生的。从 1981 年 BIOS 在 IBM 的第一台个人计算机上出现到今天,BIOS 固件系统的设计从未考虑过安全问题,导致 BIOS 系统面对的安全威胁逐步增加。本节归纳出导致 BIOS 安全风险形成的三个主要因素。

(1) 固件 BIOS 功能的扩展和增强,导致可被恶意利用的固件 BIOS 安全漏洞逐渐增加。由 BIOS OEM 厂商定制开发或其他第三方开发的 BIOS 功能模块,由于硬件或软件兼容性问题,可能造成计算机 BIOS 系统存在某种功能障碍或对计算机数据造成一定程度的破坏;BIOS 配置项的不合理或配置疏忽也可能会被攻击者利用以实现对计算机实施本地或远程的存取和控制。新一代 EFI BIOS 产品内置的 TCP/IP 协议支持、USB 协议支持、命令 Shell 支持等,使得 BIOS 更像是一个 Mini OS,在 BIOS 系统功能增强的同时也带来更多负面的安全风险。

(2) BIOS 存储芯片可采用纯软件方式进行读写所带来的安全漏洞和风险。存在多种应用需求要求能够采用纯软件方式对 BIOS 进行读写,例如:BIOS 需要与操作系统进行交互并在 BIOS flash 中存储动态配置数据;通过网络对 BIOS 固件进行更新升级以修补 CPU 或 BIOS 系统存在的缺陷,或是使得升级后的 BIOS 系统能够支持新的硬件设备。从 2000 年以后,市场上大多数主板允许计算机用户通过网络对 BIOS 在线升级。即插即用(Plug and Play,PnP)[53]协议也要求 BIOS 在运行过程中能够自动侦测硬件的变化,并将相应信息保存下来以备计算机下一次启动使用,这些信息往往需要保存到 BIOS flash 的 NVM 区域。CPU 的微码更新也要求能够方便地

对主板上的BIOS进行重写。为适应这些变化，主机板BIOS芯片由最初使用的ROM、PROM、EPROM等芯片逐步被可以使用纯软件方式改写更新的flash芯片取代。即在操作系统环境下，采用软件工具，不必使用其他辅助硬件设备，就能够对主板上存储BIOS的芯片进行读写，擦除或改写其内容。攻击者利用BIOS存储芯片的这种特性，可以向BIOS中写入恶意控制代码或篡改BIOS，从而达到控制或破坏计算机和网络系统的目的。

(3) BIOS存储芯片容量的不断扩展带来的安全风险。伴随BIOS功能扩展，BIOS存储芯片从最早的32KB逐步增加到64KB、128KB，目前512KB和1024KB的Flash存储芯片逐渐占据了BIOS存储芯片市场的主要份额。本文研究过程中对市场上计算机BIOS的抽样调查表明，存储BIOS的flash芯片其空间并未被完全占用，一般尚有几十KB甚至几百KB的剩余空间。BIOS中这些剩余空间为恶意代码植入BIOS中提供了存储便利。

这里解释一下CPU微码更新的概念。从Pentium Pro处理器开始，Intel CPU具备一项称为“微码更新”(Microcode Update)的功能，可对CPU某些缺陷进行修复。若某一批CPU集成电路在设计时有未被检测到的缺陷(Bug)时，只要这个错误还可以由微码的方式加以修正，则BIOS可以通过微码更新的功能，将Intel所送来的某批号的修正微码，在BIOS的检测运行过程中写入Pentium Pro内部的MicroROM中，这样就可以修正CPU的缺陷，而Intel也就不需要大费周折地回收存在BUG的CPU了。微码更新主要是通过指令cpuid首先读出该CPU相应的标识，若该标识与Intel提供的微码表中标识相同，则表明需要进行修复，通过wrmsr指令将微码写入CPU的微码存储区中，从而修复CPU缺陷。

4.3　BIOS安全漏洞分析

同操作系统、数据库、网络系统中已发现的大量漏洞相比较，BIOS存在的安全漏洞一直被忽视，其被发现的漏洞数量迄今为止仍然处于一个较低水平。这其中的原

因，一是对 BIOS 安全的研究起步较晚，关注者较少，二是 BIOS 系统处于底层，对其攻击或利用其攻击系统技术难度较高。而伴随 BIOS 技术和功能的进一步发展，BIOS 存在的安全漏洞及其安全重要地位正逐渐为安全业界所认知。

CVE 包含两种不同状态的安全漏洞：正式（Entry）状态和候选（Candidate）状态。正式状态的安全漏洞是已被 CVE 编辑委员会（CVE Editorial Board）审核认可并正式发布公认的安全漏洞，候选状态的安全漏洞则需要等待 CVE 编辑委员会审核认可后，才有可能被接受为正式状态。本书对 CVE 研究统计的结果显示，截止到 2007 年 12 月 14 日，最新发布的 CVE 列表（CVE List）中包含 29548 个安全漏洞。其中，正式状态的安全漏洞 3054 个，候选状态的安全漏洞 26494 个。正式状态的安全漏洞中与 BIOS 相关的安全漏洞为零个，候选状态的安全漏洞中，与 BIOS 相关的安全漏洞有 4 个。

安全漏洞的发现，是一个探索、积累和认识深入的长期过程。综合采用安全隐患情景分析[54]、手工测试、静态和动态分析等漏洞发现技术，结合 CVE 描述，本书发现并验证了 BIOS 中目前存在的多种安全漏洞，这里介绍一些主要的 BIOS 安全漏洞。

1. BIOS 远程开机漏洞

该漏洞在目前市场上的 BIOS 产品中普遍存在。BIOS Setup 中默认设置允许在计算机关闭电源而保持电源物理连接的情况下，使用特殊的工具软件，通过网卡或调制解调器打开计算机电源，远程启动计算机。这使得攻击者可能在用户毫无察觉的情况下非法访问用户计算机。

2. BIOS 定时开机漏洞

该漏洞在目前市场上的 BIOS 产品中普遍存在。BIOS Setup 中默认设置允许在计算机关闭电源而保持电源物理连接的情况下，当 BIOS Setup 中设定的日期和时间到来时，BIOS 能够自动打开电源启动计算机。这使得攻击者可能在用户毫无察觉的情况下自动处理用户预先设定的任务或者非法访问用户计算机。

3. ChipAwayVirus 漏洞

ChipAwayVirus 是由 Symantic 开发，由主板厂商集成在 BIOS 中的反引导扇区病毒模块。该模块对某些正常引导分区会错误报警，阻止系统分区或引导，特别会引起 LILO 和 Linux 的引导及安装失败，导致用户无法正常使用计算机。

4. 磁盘恢复精灵漏洞

磁盘恢复精灵是主板厂商集成在 BIOS 中的磁盘操作系统备份和恢复模块。磁盘恢复精灵模块存在设计缺陷，工作不稳定，只支持微软的操作系统文件格式，不支持 UNIX 和 Linux 的文件格式，某些情况下恢复操作可能导致用户磁盘数据被破坏，导致用户数据丢失。

5. Phoenix Net 漏洞

Phoenix Net 模块是由 BIOS 厂商 Phoenix 开发并集成在 BIOS 中的模块。该模块具备在线网络验证、网络下载上传功能。有暴露网络计算机用户个人隐私、网络行为习惯的嫌疑。

6. BIOS 弱口令漏洞

该漏洞存在于目前市场上的主流 Award 和 Phoenix 各个版本的 BIOS 产品中。BIOS Setup 口令和开机口令密码进行简单的转换后被存储，存在不同的口令密码转换后结果相同的现象，因此很容易经过有限次密码转换尝试后，找到可替代口令。而且这些 BIOS 产品，很多都设置了通用口令，如 Award 通用密码有 j256、LKWPPETER、AWARD_SW，AWI 通用密码有 AMI、BIOS、PASSWORD、AMI_SW、LKWPETER、A. M. I。这种通用口令，甚至可以看做厂商设置的后门。

7. CVE-2002-2059

该安全漏洞存在于 Intel 的 D845BG、D845HV、D845PT、D845WN 主板的 BIOS

中。在 BIOS 上电自检过程中，用户可以使用 F8 键来修改计算机引导设备(如软盘引导、硬盘引导、光驱引导、网络引导等)的引导顺序，而不受预设的超级用户密码和用户权限级别的限制。

这使得攻击者能够通过改变引导设备的顺序，在计算机上安装不同的操作系统，或改变引导分区的内容，或访问计算机硬盘中文件系统的内容。

8. CVE-2005-0963

该安全漏洞存在于东芝(Toshiba)ACPI BIOS 1.6 产品中。通常硬盘的主引导分区 MBR(Master Boot Record)最多可包含 4 个引导项，每个引导项都可能成为活动分区(Active Partition)而允许引导安装在该分区上的操作系统。而该型号的 BIOS 产品在实现中代码出现逻辑错误，导致每次计算机引导时只检查分区表中的第一个分区是否为活动分区，而忽略随后其他分区活动状态的循环检查。这样，即使是正确的分区表，也可能导致计算机系统不能正常引导启动进入操作系统。

这个安全漏洞可能会导致对计算机系统的拒绝服务攻击。

9. CVE-2005-4175

该安全漏洞存在于系微(Insyde)的 V190 BIOS 产品中。在系统进入操作系统后，BIOS 没有清除用户输入的键盘缓冲区内存中存放的 BIOS 密码，密码以明文形式仍然存在于 BIOS 数据区(BIOS Data Area，BDA)中的 0x1e 偏移处的键盘缓冲区中。

可通过直接读取物理内存的内容获取 BIOS 密码，导致 BIOS 密码泄露。

10. CVE-2005-4176

该安全漏洞存在于 AWARD BIOS Modular 4.50pg 产品中。CVE-2005-4176 同 CVE-2005-4175 是同一漏洞现象，只是存在于不同的 BIOS 产品中。

4.4 固件 BIOS 安全威胁分析

本书对 BIOS 的安全威胁进行分类研究，通过剖析 3 起典型的 BIOS 安全威胁作用机制和原理，以期对固件 BIOS 的安全保护起到借鉴、参考和启发作用。

4.4.1 固件 BIOS 安全威胁的分类

本书将 BIOS 安全威胁按照不同的来源分为两类：内部威胁和外来威胁。内部威胁来自 BIOS 自身，是由于 BIOS 自身扩充功能设计障碍导致本地计算机硬件、磁盘数据或系统软件造成损害，或配置项目不当被攻击者利用，或编程不严谨造成信息泄露等；外来威胁来自 BIOS 外部，主要是外部攻击者向 BIOS 存储芯片中植入的恶意代码，或向 BIOS 固件发动物理攻击导致 BIOS 系统拒绝服务等。

按照 BIOS 安全威胁作用机制的不同，又将 BIOS 安全威胁进一步归纳为 5 种类型。

1. BIOS 功能障碍

BIOS 厂商或主板厂商在 BIOS 开发过程中，由于逻辑不严谨造成 BIOS 提供的功能在特定情况下出现障碍，对计算机系统或数据造成一定的破坏。如集成在 BIOS 中的 ChipAwayVirus 防引导扇区病毒模块会错误识别分区信息，导致 Linux 操作系统装载软件失败；磁盘恢复精灵模块在某些情况下会造成硬盘恢复失败和数据丢失；Toshiba ACPI BIOS 1.6 产品在检查磁盘活动分区时只检查 MBR 的第一个分区目录项，导致许多正确的 MBR 分区不能正常启动。

2. BIOS 配置漏洞

用户本地计算机 BIOS 配置不合理，导致攻击者可以通过网络远程对用户本地计算机的某些 BIOS 选项重新设置，进而配合使用工具软件完成对本地计算机的远程存

取和控制。典型的配置漏洞包括允许对计算机远程开机、唤醒、允许擦写 BIOS Boot Block 区等。

3. BIOS 信息泄露

存储在 CMOS 中的 BIOS 配置信息没有受到任何保护，通过 I/O 端口可以读写这些配置数据，可能被恶意者利用这些数据进行主机攻击或控制。例如：Award BIOS 在引导操作系统后，并没有清除内存缓冲区中的 BIOS 密码信息，恶意者通过读取物理内存可以获取计算机 BIOS Setup 密码和开机密码。

4. BIOS 物理攻击

攻击者使用软件工具直接改写或擦除 flash 芯片存储的内容导致计算机不能正常启动，从而形成对 BIOS 系统的拒绝服务攻击。CIH 病毒是这种物理攻击的典型案例。在微软系列操作系统以及 Linux 操作系统环境下，都存在这样一些专用或通用的 BIOS flash 读写工具软件，如 Awdflash、WinFlash、uniflash 等。

5. BIOS 恶意代码

存储 BIOS 系统的 flash 芯片可以在操作系统环境下使用软件方式进行读写操作，而 flash 芯片中往往又会空余几十 KB 到几百 KB 的空间，恶意攻击者利用这两个条件就可以将经过包装的恶意代码植入到存储 BIOS 的 Flash 芯片中，并通过合适的机制使得嵌入在 BIOS 中的恶意代码能够在 BIOS 或操作系统运行过程中获取对系统的控制权。

BIOS 功能障碍、配置漏洞、信息泄露属于 BIOS 内部威胁，而 BIOS 物理攻击、BIOS 恶意代码则属于外来威胁。

4.4.2 CIH 病毒对固件 BIOS 破坏分析

早在 1998 年 7 月 26 日，CIH 恶性病毒就已经在美国开始大面积肆虐。1998 年

8 月 26 日，该病毒入侵中国。1999 年 4 月 26 日，CIH 全球大面积爆发，导致大量的 PC 停止工作，用户数据遭到严重破坏。

CIH 病毒发作时其中的一种破坏行为是直接向计算机主板固件 BIOS 芯片中和硬盘中写乱码，破坏力非常大，可造成主机无法启动，硬盘数据全部被清洗，大量重要资料无法复原，严重者甚至连计算机主板硬件也不得不更换。

CIH 病毒在当时是一种全新概念的病毒。本节重点分析 CIH 病毒对固件 BIOS 攻击的原理机制。

1. CIH 攻击固件 BIOS 的机制

CIH 病毒首先在 Windows 系统下获取 RING0 级特权，使得病毒代码能够获取直接对物理设备和端口进行操作的权限。病毒破坏时，通过主板的 BIOS 端口地址 0CFEH 和 0CFDH 向 BIOS 引导块(boot block)内各写入一个字节的乱码，乱码直接导致引导代码执行错误，使计算机主机无法继续完成启动过程。其后 CIH 病毒的变种，更是变本加厉地用垃圾信息去填充整个固件 BIOS 的内容，造成 BIOS 代码和数据的彻底破坏。由于这种破坏发生在固件 BIOS 芯片内部，因此要想恢复，必须使用专业方法，取下主板上的 BIOS 芯片，使用硬件设备重写主板上的 BIOS 芯片中的代码和数据。

CIH 病毒对 BIOS 的攻击采用的是通过直接写端口技术实现的。BIOS 采用了 flash 存储芯片，flash 存储芯片允许在一定的电压下，利用软件方式从 BIOS 端口中读出和写入数据，以便进行 BIOS 程序的升级更新。CIH 病毒正是利用闪存的这一特性，往 BIOS 芯片里写入乱码，造成 BIOS 芯片中的原有代码内容被彻底破坏，导致计算机无法启动。

2. CIH 攻击固件 BIOS 原理和制约因素

分析 CIH 病毒对 BIOS 的破坏机制，可以发现，这种病毒只能对部分特定的 BIOS flash 进行破坏。其破坏技术受到两个方面因素的制约：BIOS flash 芯片的改写类型和读写端口地址。

个人计算机上用来保存 BIOS 程序的 flash ROM 擦写技术可以分成两种类型。

(1) 建立在 Intel 公司的一项专利技术 Boot Block 保护概念基础之上的擦写技术,使用此类技术的一般都是 Intel 公司出的 flash ROM 芯片,如常见的 Intel 28F0 芯片。这种 BIOS flash 芯片通常包含两个电压接口,其中 +12V 一般用于 Boot Block 的改写。Boot Block 为一特殊的区块,它主要用于保存一个最小的 BIOS 用以启动最基本的系统之用。当 flash ROM 中的其他区块内的数据被破坏时,只要 Boot Block 内的程序还处于可用状态,则可以利用这一基本的 BIOS 程序来启动一个最小化的系统。一般情况下,起码应当支持软盘的读写以及键盘的输入,这样就有机会使用软盘来重新构建整个 flash ROM 中的数据。一般的主板上均包含有一个专门的跳线,用来确定是否给此 flash ROM 芯片提供+12V 电压,只有需要修改 flash ROM 中 Boot Block 区域内的数据时,才需要短接此跳线,以提供+12V 电压。另外一路电压为+5V 电压,它可以用于维持芯片工作,同时为更新 flash ROM 中非 Boot Block 区域提供写入电压。也就是说,主板上的+12V 跳线是为了防止更新 flash ROM 中的 Boot Block 区域而设置的,如果想升级 BIOS,而此升级程序只需要更新 Boot Block 区域以外的 BIOS 程序,则主板上的跳线根本没必要去跳,因为更新 Boot Block 区域以外的数据并不需要+12V 电压。这样,即使升级失败,也还存在着一个 Boot Block 中的最基本 BIOS 可以使用,这样就可以使用软盘来恢复原先的 BIOS 映像(一般在升级的时候,都会提示用户保存当前的 BIOS 内容映像)。

如果用户不小心使主板上的跳线处于短接状态,即 flash ROM 芯片已经有+12V 电压了,则这时候 Boot Block 内容也是可以采用软件方式改写的。

(2) 无 Boot Block 保护的 flash Rom,其芯片所有内容在+5V 的单一电压下就可以进行改写,如 Atmel、SST、Winbond 等公司生产的 flash 芯片。

可见,如果主板上存在 BIOS flash 芯片的保护跳线,则病毒只能对 Boot Block 之外的其他 BIOS 内容造成损坏,否则病毒就可以对 BIOS 的全部内容造成损坏。

即使如此,CIH 病毒也只能对少数类型的主板上的 flash BIOS 芯片构成威胁。这是因为,CIH 采用直接读写端口地址技术改写 BIOS 芯片内容,而不同主板的 BIOS 端口地址各不相同。CIH 病毒受到自身代码尺寸的限制,不可能智能检测

BIOS flash 类型并包含所有种类的 flash 芯片端口地址数据。另外，这些端口地址往往是商业技术秘密，只在少数芯片组厂商、BIOS 厂商、主板厂商之间共享，其他人很难了解到这些数据。

以上分析，可见 CIH 对 BIOS 固件的攻击，主要是以 flash 芯片替换原来不可软件改写的 ROM 芯片导致的结果。但 flash 芯片的使用，既有商业利益的驱使，也有科学技术发展和 BIOS 自身需求的驱使。倒退到原来的不可软件改写的 ROM 已是不可能。要保护 BIOS 固件的安全，只能探索新的途径。

4.4.3　PhoenixNet 分析

自 1999 年始，美国 BIOS 厂商 Phoenix 公司对其 BIOS 产品实施了一个称作凤凰网(PhoenixNet)的项目计划[55]。该项目通过在 BIOS 固件中嵌入一个代码模块 ILS (Internet Launch System)，试图帮助用户更方便地使用凤凰网提供的服务，如为用户自动侦测网络连接、启动 Web 服务、自动下载服务软件包并安装执行等。

本节对包含 PhoenixNet 的 BIOS 固件产品进行了实验研究。实验中采用的相关产品型号如下。

主板：VIA694ProB(DMA100)

BIOS 产品时间：10/31/00

BIOS 产品类型：Phoenix Award Modular BIOS v6.00PGN

BIOS 产品序列号：10/31/2000-694X-686B-6A6LJL1BC-00

操作系统：Windows 98、Windows 2000 Professional、Windows XP

实验现象表明，PhoenixNet 的 ILS 在 Windows 98 操作系统下才会被激活，而在 Windows 2000 和 Windows XP 操作系统下则不会被激活使用。

实验结果表明，PhoenixNet 的 BIOS 固件在操作系统运行后，能够自动执行 BIOS 芯片中的一个代理程序，该程序向远端服务器(服务器域名为 www.seqdl.com)上传 BIOS 固件产品验证信息，通过验证后从服务器下载一个 seqinstall.exe 程序到本地计算机上并执行该程序。

进一步对PhoenixNet的BIOS产品进行剖析可以发现，嵌入在Phoenix BIOS产品中的ILS是一个大小为87KB的独立BIOS程序模块，模块名为rosupd. rom。使用Phoenix公司提供的BIOS映像文件编辑工具cbrom. exe可以从BIOS映像文件中提取或删除该模块，也可将提取出的该模块嵌入到其他不包含该模块的Phoenix BIOS产品中，从而使该产品具备PhoenixNet功能。

实验中通过对PhoenixNet安装前后硬盘文件系统变化的跟踪比较可以发现，BIOS中的ILS模块在Windows 98系统初次安装后，就从BIOS flash中释放到Windows系统目录\WINDOWS\SYSTEM\下。释放后的内容包含5个文件：PTLSEQ. CPL、PTLSEQ. DAT、PTLSEQ. MET、PTLSEQ. RCL、PTLSEQ. REP，执行的入口点是一个Windows控制面板程序PTLSEQ. CPL。

进一步研究分析BIOS中的ILS模块可以发现，该模块是一个压缩模块，没有任何可以在BIOS运行阶段直接执行的代码。因此可以判断，flash芯片中的ILS模块向操作系统(或硬盘)中释放并自动执行的过程，是由操作系统主动完成的，即由操作系统把该模块从BIOS中“拉”入到操作系统(或硬盘中)。此处将这种嵌入在BIOS中的操作系统程序释放到操作系统(或硬盘中)的技术称为“拉(pull)”技术，以便同本书后面提出的“推(push)”技术相区分。而由于Windows 2000和Windows XP不再支持对PhoenixNet的pull，因此嵌入有PhoenixNet模块的BIOS产品在安装这两种操作系统时不再生效。

尽管Phoenix公司声称PhoenixNet计划是为了向计算机用户提供更多更方便的网络服务，但由于嵌入在BIOS中的ILS模块具有远程安装、控制、回传、定位等功能，遭到广泛的抵制和批评，Phoenix公司最终被迫终止了PhoenixNet项目。

PhoenixNet给出的新的安全启示是：BIOS固件层的代码或代理程序(Agent)，可用于实施对操作系统自身或操作系统层的系统保护或攻击。

4.4.4 ACPI BIOS rootkit 和 PCI rootkit 分析

著名的信息安全机构SANS在文献[56]中将rootkit定义为：rootkit是攻击者

用来隐藏自己的踪迹和保留计算机或网络系统的管理员级 root 访问权限的工具。入侵者入侵后往往会进行清理脚印和留后门等工作，最常使用的后门创建工具就是 rootkit。它是入侵者在入侵了一台主机后，用来做创建后门并加以伪装用的程序包，这个程序包里通常包括了日志清理器、后门等程序。rootkit 通常存储在系统硬盘上，混杂于操作系统、应用程序等文件中，难于检测，难以清除。

2006 年 Black Hat 会议上，英国 Next-Generation 安全软件公司首席安全顾问 John Heasman 阐述了一种新的 rootkit 技术[26]。按照 John Heasman 的方法，黑客可以利用“高级配置和电源管理接口”(ACPI)及其编程语言 ASL(ACPI Source Language)/AML(ACPI Machine Language)在主板 BIOS 闪存芯片中隐藏 rootkit 恶意代码，黑客甚至能够利用自己编写的恶意功能取代 ACPI 中的正常功能。这类固件 BIOS rootkit 攻击的危害是：系统重启对它没有任何作用，在硬盘上无法探测到它，即使重新格式化硬盘或重新安装操作系统也不会对它有任何影响。

2007 年 Black Hat 会议上，John Heasman 继续阐述其对固件 rootkit 技术的研究成果，这一次他将 rootkit 技术应用到了主板 PCI 板卡的 OPROM 闪存中[27]。利用部分 PCI 扩展卡上的 flash ROM 允许使用软件进行改写的特点，在 PCI 扩展卡上 flash ROM 中嵌入 rootkit 代码。当 BIOS 运行时，会自动搜索发现 PCI 设备，并将所发现的 PCI 扩展 ROM 内容拷贝到内存 0XC0000-0xC7FFF(视频 ROM)或 0xC8000-0xEFFFF(非视频 ROM)处，从扩展 ROM 内容第 3 个字节处开始执行扩展 ROM 代码。标准的 Option ROM 代码格式为：

```
Signature db 55h,0aah          ;ROM 标志
Length    db ?                 ;ROM 长度,以 512 字节为单位
Jmp near ptr Start             ;入口代码
Reserved db 20 dup(0)
Start:                         ;代码从这里开始
```

下面重点分析 ACPI BIOS rootkit 作用机制和原理。

有必要首先分析一下系统 BIOS 中 ACPI 相关内容与操作系统中 ACPI 相关内容之间的互作用机制，ACPI BIOS rootkit 正是利用这种互作用机制实现其隐藏于

BIOS 中，而执行于操作系统环境下。

1. ACPI 表（ACPI Table）

ACPI 表是兼容 ACPI 标准的操作系统和系统固件 BIOS 之间的主要接口。跟硬件配置和电源管理相关的系统信息、特性和控制方法都存储 ACPI 表中，表中的内容根据需要由 BIOS 代码动态收集或厂商静态指定存储。

2. OSPM 模块

在符合 ACPI 规范的操作系统中，有一个模块被称为 OSPM（Operating System-directed configuration and Power Management）。该模块是操作系统中处理 ACPI 的核心模块，负责提供对 ACPI 定义的属性和方法的支持，包括使用固件 BIOS 提供的接口在内存中定位 ACPI 表、解释执行 AML 代码、利用 ACPI 表中的内容列举和配置主板上的设备等。在 BIOS 固件中隐藏的 ACPI rootkit 就是由操作系统的 OSPM 解释执行的。

通过 ACPI 表，OSPM 可以透明地控制系统设备，而无需知道设备控制实现的细节。

ACPI 表实际上是多个表的链结构的集合，保存在系统 RAM 空间，其内容在操作系统运行期间一直被保留。其中，根系统描述指针 RSDP（Root System Description Pointer）结构是所有其他表的目录结构，是 OSPM 访问 ACPI 表的入口，如图 4.1 所示[50]。

对于传统 BIOS，OSPM 通过在 0E0000h 到 0FFFFFh 之间以 16 字节对齐方式搜索 RSDP 结构的标志字符串 RSD PTR 来获取 ACPI 表的入口，通过 INT 15h 的 E820h 功能确定 ACPI 表占用的系统保留内存空间[50]；对于 EFI BIOS，OSPM 通过 EFI 系统表（EFI System Table）获取 ACPI 表的入口，通过 EFI 定义的标准引导服务（boot services）GetMemoryMap( ) 确定 ACPI 表占用的系统保留内存空间[7,50,51,58]。

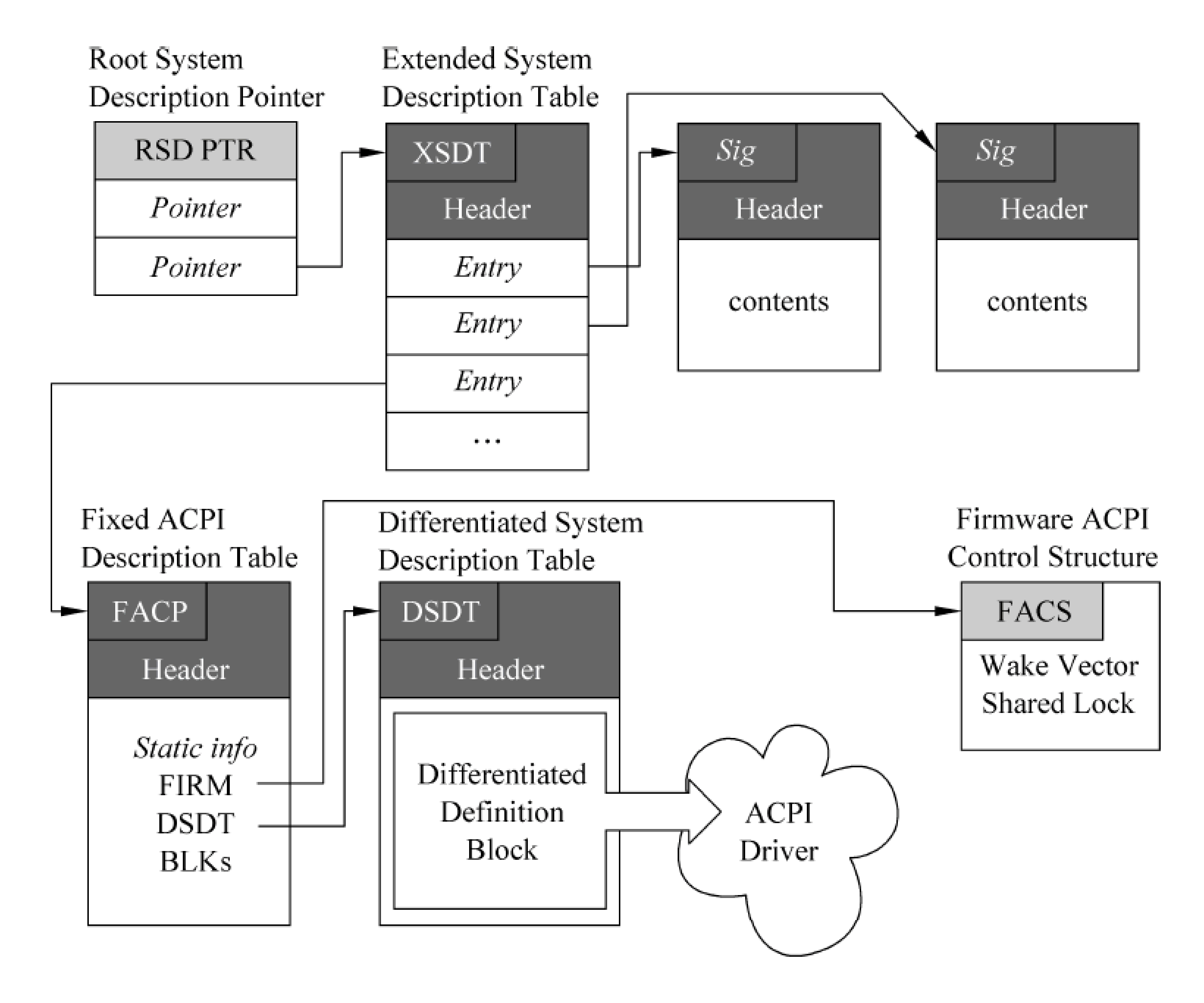


图 4.1　ACPI 表

3. ASL/AML 语言

ASL(ACPI Source Language)语言是用于定义 ACPI 对象及其控制方法的源语言，通过编译器(例如微软的 ASL 编译器)编译生成 AML(ACPI Machine Language)代码。

ASL 属于高级编程工具语言，代码编写容易，而且具备强大的访问能力，如直接访问物理 I/O 端口、系统内存、PCI 配置寄存器、CMOS 内容等。

John Heasman 演示的 ACPI BIOS rootkit 正是利用了上述的 APCI 特性，通过使用 ASL 语言编写一个 rootkit，或在 ASL 代码中包含精心设计的 shellcode，将这样的 rootkit 存储或嵌入到 BIOS 芯片中。当操作系统运行后，OSPM 会通过 ACPI 表自动解释执行从 BIOS 中释放到内存中的 ACPI rootkit 代码。

当然，ACPI rootkit 能够成功的前提，是攻击者有机会并且有能力将 ACPI rootkit 嵌入到 BIOS flash 中。这个问题在 John Heasman 的报告中没有涉及。本书后续 4.6 节内容将对这一问题进行阐述。

4.5 操作系统对 BIOS 固件服务的引用研究

固件 BIOS 代码的运行要早于操作系统代码的运行，其运行路径为：

BIOS —> OS Loader —> OS

在操作系统运行后，BIOS 代码的运行就告一段落。但是，操作系统运行起来后，并不是同 BIOS 代码完全脱离开，而是存在某些运行依赖关系。研究 BIOS 同操作系统的引用交互机制，对于探索通过 BIOS 对操作系统提供保护的安全机制，或者在 BIOS 和操作系统间建立安全通道，阻止来自系统底层或上层的安全威胁，从而保护计算机系统的整体安全，是一条重要的路径。

总结起来，固件 BIOS 为操作系统提供的服务分为代码和数据两部分，从这两部分入手，两者之间通常包括以下 4 种代码和数据交互的方式。

（1）**BIOS 中断（BIOS Interrupts）** 固件 BIOS 中提供了大量的中断代码程序，这些中断程序为操作系统和应用程序提供了一种标准的功能代码调用方式。因此，无论固件类型和操作系统类型如何，中断程序始终是固件 BIOS 为操作系统提供代码服务的最主要形式。

（2）**BIOS 32 服务（BIOS 32 Services）** 传统 BIOS 中为 32 位的操作系统和设备驱动提供 BIOS 层的 32 位代码服务。虽然形成了一些规范[59]，但很快被淘汰，在大多数 BIOS 产品中未提供这类服务，一些典型的商业操作系统，如 Windows 2000、Windows XP、Linux 等在其运行过程中也都不再调用这类 BIOS 32 服务。

（3）**ACPI 接口** 这是 BIOS 与 OS 间交互的一种重要途径，ACPI 既可以为操作系统和固件之间提供代码交互，也可以提供动态数据方式。

（4）**SMBIOS 接口** SMBIOS 内容中包含固件和操作系统之间的动态交互数据，以及为获取 SMBIOS 接口信息/结构而提供的固件接口程序，但更多的，这种交互方式是一种 BIOS 固件向操作系统单向的数据交互。

在新一代 EFI 固件和较新的操作系统产品中，对于传统 BIOS 中断程序来讲，其使用和地位日益下降，只是为了保持向前的兼容性才会提供(如 EFI 固件中的 CSM 模块，就专为提供传统 BIOS 固件中断程序服务而开发)。而 ACPI 接口，以及 EFI 中新定义的固件服务，才是后续固件系统与操作系统之间交互的主流方式。

4.5.1　BIOS 中断概述

对于 80x86 系统，允许最多存在 256 种中断，中断类型号为 00h～0FFh。由硬件设备产生物理信号引发的中断称为硬中断，否则称为软中断。在内存的最低 1KB 空间中(00000h～000003FFh)存放着每种中断的入口地址(也称为中断向量，即中断服务程序的起始地址)，该地址指向中断程序的第一条可执行指令。存放中断向量的低 1KB 内存空间称为中断向量表。每个中断用一个中断类型号来代表，中断类型号 N 和其对应的中断向量在中断向量表中地址 Addr 的对应关系为：Addr ＝ 4 * N。

并非所有 256 种中断都需要实现，BIOS 只根据系统运行需要实现其中的一部分，其他中断可由软硬件厂商或用户扩充实现。表 4.1 列出了 Phoenix BIOS 4.0 Revision 6.0 产品所实现的 BIOS 中断[60]。

表 4.1　Phoenix BIOS 4.0 实现的中断

中断类型号(h)	中断向量表地址(h)	中 断 描 述
02	8	不可屏蔽中断
05	14	屏幕打印中断
06	18	286 LoadAll 句柄
08	20	IRQ0——系统定时器中断
09	24	IRQ1——键盘中断
0B	2C	IRQ3——COM2 中断
0C	30	IRQ4——COM1 中断
0D	34	IRQ5——LPT2 中断
0E	38	IRQ6——软盘中断
0F	3C	IRQ7——LPT1 中断

续表

中断类型号(h)	中断向量表地址(h)	中 断 描 述
10	40	BIOS 视频接口
11	44	BIOS 设备检查
12	48	BIOS 内存请求
13	4C	BIOS 磁盘接口
14	50	BIOS 串行接口
15	54	BIOS 系统功能接口
16	58	BIOS 键盘接口
17	5C	BIOS 并行打印机接口
18	60	BIOS 次引导请求
19	64	BIOS 主引导请求
1A	68	BIOS 系统定时器接口
1B	6C	BIOS 的 CTRL+Break 中断
1C	70	BIOS 用户系统定时器中断
1D	74	BIOS 视频初始化参数表
1E	78	BIOS 磁盘参数表
1F	7C	BIOS 视频图形字符
40	100	BIOS 磁盘
41	104	BIOS 固定磁盘 0 参数表
46	118	BIOS 固定磁盘 1 参数表
70	1C0	IRQ8——实时时钟中断
71	1C4	IRQ9——IRQ2 重定向
74	1D0	IRQ12——PS2 鼠标中断
75	1D4	IRQ13——数学协处理器中断
76	1D8	IRQ14——主 IDE 硬盘
77	1DC	IRQ15——从 IDE 硬盘

从表 4.1 可以看出，Phoenix BIOS 实现了 256 种中断中的 35 个中断，这些中断既有响应硬件设备初始化/操作的硬中断，也有为系统/用户调用接口实现的软中断。其他传统 BIOS 产品所实现的中断与表 4.1 所列的中断大同小异。选择哪些中断的实现，并没有一个行业的标准或规范对此有所规定，其中最大的影响因素，一是硬件设备的兼容需求，二是软件系统、特别是操作系统兼容需求的发展。

4.5.2 Windows XP/2000 运行依赖的 BIOS 中断

不同的操作系统对 BIOS 提供的中断使用和依赖程度是不一样的。早期 DOS 系统,其实现严重依赖于 BIOS,包括 DOS 中断 int 21h 等操作系统高级中断都是通过调用底层 BIOS 中断来实现的。随着操作系统的发展,在操作系统层往往会用一些执行效率更好、更能有效发挥硬件性能、更完善的高层驱动来代替底层 BIOS 驱动和中断。但并非所有的 BIOS 中断在操作系统层都可以替代,试验证明,现有的操作系统,在其运行阶段,或多或少都会调用 BIOS 层提供的中断程序,当然其保护模式下实现的中断机制会不一样[39,40,41]。表 4.2 列出了 Windows XP/2000 系统在运行阶段对 BIOS 中断调用的统计,该数据根据马里兰信息系统安全试验室的 Adam Sulmicki 的研究报告[9]整理而来。

表 4.2 Windows XP/2000 对 BIOS 中断调用统计

中断号(h)	中断描述	中断调用次数(包括功能号重复调用)		中断调用次数(删除功能号重复调用)	
		Windows 2000	Windows XP	Windows 2000	Windows XP
10	BIOS 视频接口	13	11	12	11
11	设备检查	1	1	1	1
13	磁盘接口	Lots	Lots	6	5
15	系统功能接口	~22	22	~4	8
16	键盘接口	12	5	2	2
1A	系统定时器接口	12	13	3	3

由此可见,操作系统启动运行后,仍然要依赖于 BIOS 提供的某些中断功能,而非完全使用操作系统自身的代码来替换这些中断。这些被操作系统依赖的中断,构成了 BIOS 与操作系统之间代码执行的一个通道。如果破坏这个执行通道,或者篡改这个通道中的部分代码,如采用中断钩子(Hook)技术,将可能引起操作系统的崩溃、安全失效或不可信任。

4.6 一种新型固件 BIOS 木马

“木马”的全称是“特洛伊木马(Trojan Horse)”,通常指隐藏在操作系统、磁盘中实施远程监控和信息窃取的恶意程序。通过对固件 BIOS 安全威胁的分析,本文提出固件 BIOS 木马的概念,作为一个例证,实现了一种固件 BIOS 木马程序。

BIOS 木马是指隐藏存储在 BIOS flash 芯片中的木马程序。普通木马隐藏存储在硬盘中,而 BIOS 木马隐藏存储在硬件芯片中。BIOS 木马既能实现普通木马所能实现的功能,又具备普通木马所没有的优势:BIOS 木马不受计算机重装操作系统的影响、不受计算机硬盘格式化的影响、不受计算机更换硬盘的影响、不易被现有的查杀毒软件检测和清除等。BIOS 木马一旦被种植进入 BIOS 中,只能通过对 BIOS 固件专门消毒才能被清除。因为 BIOS 木马具有这种一劳永逸的植入特点,BIOS 木马正受到众多黑客的青睐。但由于 BIOS 木马实现上的技术难度和局限性,这种木马目前只在小范围内使用和存在。

BIOS 木马技术可进一步发展成 BIOS Agent 技术,通过在 BIOS 中嵌入代理程序,利用 BIOS 处于更底层、与硬件结合更紧密的特点,提供对计算机系统更为可靠的定位、监控、安全保护等功能。

4.6.1 固件 BIOS 木马的封装

木马需要封装成符合格式要求的标准 BIOS 程序模块,并且加入使得木马能够被激活的“推” 或“拉”机制,才能够使得隐藏在 BIOS 中的木马模块能够被激活并在操作系统运行过程中实现监控和信息窃取行为。

BIOS 木马模块封装采用 Shell-Stone 两层结构。外层的 Shell 是符合格式的 BIOS 模块程序,内层的 Stone 是一个普通的木马程序。Shell 的主要作用是在 BIOS 执行的结束阶段获取系统控制运行权,将自身包容的 Stone“推”进操作系统/硬

盘中。

市场上现有的不同固件 BIOS 产品模块程序结构各不相同，没有统一的格式标准。通过逆向工程的方法，本书获取到如图 4.2 和图 4.3 的 Phoenix 4.0 和 Award 6.0 BIOS 产品的一种可执行模块程序的结构。其共同特点是代码和数据都在同一个段中，段被限制为 64KB 大小。

标识：	DB '$INI&'
入口代码：	PUSHAD
模块代码和数据：	……
返回代码：	POPAD RETF

图 4.2　Phoenix 4.0 BIOS 可执行模块程序结构

标识：	DB 55h, 0aah
长度：	DB?
入口代码：	PUSHAD
模块代码和数据：	……
返回代码：	POPAD RETF

图 4.3　Award 6.0 可执行模块程序结构

BIOS 木马外层的 Shell 就是如图 4.2 和图 4.3 所示结构的一个模块程序，而 BIOS 木马的 Stone，则以数据的形式包含在 Shell 模块的数据部分。当 Shell 获取系统控制权被运行时，会将包含的 Stone“推”出，然后结束自己的运行，将系统控制权交还给 BIOS 自身的代码，继续完成操作系统的装载引导过程。

4.6.2 固件 BIOS 木马的植入

黑客在利用系统漏洞或用户疏忽侵入系统后，通常将木马伪装成系统服务或文件，或者将木马同其他文件捆绑在一起，实现木马的隐藏存储。BIOS 木马则不同，黑客需要将木马写入到 BIOS flash 芯片中。

BIOS 木马的植入包含三个步骤：

（1）从 BIOS flash 芯片中读出 BIOS 内容形成映像文件 bios.rom。映像文件长度通常和 BIOS flash 芯片容量一致，典型的如 256KB、512KB 或 1024KB。

（2）计算 BIOS 映像文件中的空白区大小，将封装好的 BIOS 木马模块嵌入到 BIOS 映像文件中空白区中。

（3）将嵌入木马模块的 BIOS 映像文件 bios.rom 重新写入到 BIOS flash 芯片中。

模块程序必须按照对应固件产品的可执行模块结构封装好，并且嵌入到固件 BIOS 映像文件中后，要保证该模块在 BIOS 运行过程中能够被调用执行。调用执行模块的机制，随各种固件产品的不同而有较大的差异，如 Award 固件产品其模块执行顺序就是模块顺序存放的顺序，而 Phoenix 固件产品则是将模块链接成一个双向链表，按照链表中模块的链接顺序来执行。因此，模块写入固件映像文件中的空白区后，还要修改相应的链表结构，使模块能在正确的位置，合适的阶段被执行。这里不再一一赘述。

这里重点讨论软件实现 BIOS flash 芯片读写的方法。在操作系统下，有 3 种方式可实现对 BIOS flash 芯片内容的读写操作：

- 针对 flash 芯片进行操作，这需要从 flash 芯片厂商获取 flash 芯片的 datasheet 资料[61]；
- 打开南桥中的通道，直接对 4GB 物理内存的最高端进行读写操作，因为固件 BIOS 内容被映射到了这一块高端内存，打开南桥中的开关后，对最高端物理内存的读写操作，实际上就是对 BIOS flash 芯片的读写操作；

- 调用 BIOS 中提供的 SMI(System Management Interrupt)flash 功能，Award 和 AMI BIOS 厂商在其 BIOS 中都提供了对 BIOS 内容进行读写操作的接口，Phoenix BIOS 则没有提供该种操作接口。

当然，在保护模式下的操作系统中要想实现对 BIOS flash 芯片的操作，需要通过编写驱动程序的方式来实现。BIOS 厂商通常会提供 BIOS flash 工具帮助用户升级和更新 BIOS，如 Award BIOS 的 Awardflash、winflash，Phoenix BIOS 的 winphlash 等。

开源软件 uniflash 则是一个兼容性很强、包含大多数 flash 芯片支持的 BIOS flash 工具。

4.6.3　固件 BIOS 木马的激活

BIOS 木马的激活，是指使得隐藏存储在 BIOS flash 芯片中的木马，在 BIOS 运行过程结束、将系统控制权移交给操作系统，并且在操作系统成功运行后，BIOS 木马能够重新获取系统控制权，运行在操作系统环境下，完成木马应用的功能。

本书将 BIOS 木马的激活技术分为两类：推(Push)和拉(Pull)。推技术是指在 BIOS 运行阶段，由 BIOS 木马的 Shell 程序将木马程序 Stone 释放成操作系统环境下的可执行程序并且修改操作系统配置变量，使得该木马程序能够在操作系统启动后自动被执行。拉技术是指 BIOS 木马的 Shell 程序将木马程序 Stone 常驻在内存中，并将其设置成能够被操作系统自动扫描调用执行的驱动程序(如 ACPI 驱动、PnP 驱动或 PCI 驱动)，操作系统启动后，能够在某种触发条件满足的情况下，自动去内存中找到并调用执行木马 Stone 程序。图 4.4 演示了 BIOS 木马“推”和“拉”技术的区别。图中 Stone 到动态 Trojan 的实线箭头描述的是 Push 方式，而虚线箭头描述的是 Pull 方式。

John Heasman 的 ACPI Rootkit 是“拉”技术的典型应用，而对 int 13h 的 Hook 也是一种典型的“拉”技术。本书实现的 BIOS 木马采用了一种“推”技术，通过 BIOS 中的木马 Shell 对硬盘文件系统直接进行读写操作，将 BIOS 木马的 Stone 部分替换/

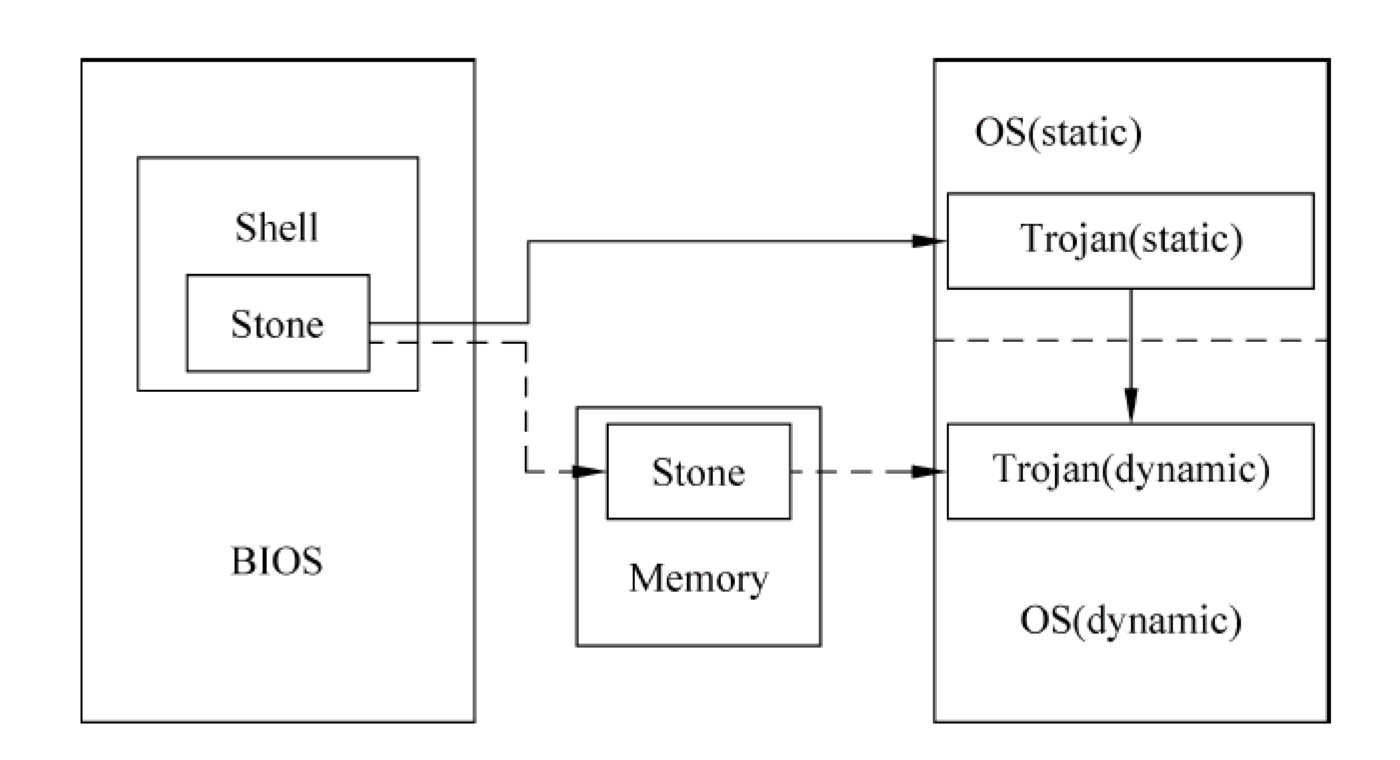


图 4.4 BIOS 木马激活的 Push/Pull 方式

伪装成硬盘中的系统服务文件，并修改操作系统启动配置，使得 Stone 木马服务在开机后能够自动运行。

如果将木马 Stone 替换成安全 Agent，则上述技术可实现基于固件对操作系统和计算机系统实施安全管理、监控、保护等应用。

4.7 本章小结

计算机固件 BIOS 承载的功能持续扩展和增强，以及新技术的发展应用需求的持续增长是固件 BIOS 安全漏洞和威胁逐步形成并呈递增趋势的主要原因。本章首次系统地指出现有传统固件 BIOS 产品存在的部分安全漏洞及其危害。将 BIOS 安全威胁按威胁来源分为内部威胁和外来威胁两大类，按作用机制分为功能障碍、配置漏洞、信息泄露、物理攻击、恶意代码五类。

本章分析归纳了固件 BIOS 与操作系统的交互性，包括数据传递和代码调用。数据传递的手段有：SMBIOS 类、ACPI 表、BDA、EBDA 等。代码调用有：ACPI 驱动、中断。这种交互性决定了固件 BIOS 与操作系统之间存在安全依赖和互动关系，这种关系可用于信息战攻防的两个方面，因此对固件 BIOS 安全的重要性和基础性应该引起足够的重视。本书提出并实现的一种新型 BIOS 木马很好地证明了这一点。



第 5 章

计算机固件BIOS安全检测方法与实现

现有的漏洞扫描软件主要用于发现主机操作系统、数据库、网络系统存在的安全漏洞和弱点，其中包括系统软件本身存在的问题，也包括用户配置存在的失误。防病毒软件则用于扫描、发现和清除主机系统感染的病毒软件，包括病毒、蠕虫、木马、流氓软件等。纵观这些技术手段和产品，其防护和检测的对象存在一个盲点：固件 BIOS。通过第 4 章的分析我们知道，在计算机 BIOS 系统中同样存在一些安全漏洞，而且针对 BIOS 系统的安全攻击和威胁呈逐年增多的趋势。由 Intel 提出并倡导的新一代 EFI BIOS(Extensible Firmware Interface)将赋予 BIOS 系统更多的功能，更重要的角色，甚至于存在一种趋势：将来的操作系统将成为运行在 BIOS 系统环境下的一个应用。因此，需要一种能够对固件 BIOS 系统进行安全检测的工具。

本章研究对计算机 BIOS 系统进行安全检测的方法和技术手段，包括 BIOS 系统的漏洞检测和 BIOS 系统中恶意代码的检测，并介绍本书开发实现的一种计算机固件 BIOS 安全检测系统。

5.1　固件 BIOS 安全检测的复杂性

第 4 章分析的结果显示，计算机固件 BIOS 系统的安全检测，应当包括两方面的内容：一是扫描和发现 BIOS 系统可能存在的安全漏洞，帮助用户针对这些漏洞采取相应的补救措施；二是检测 BIOS 系统中是否存在恶意代码，识别并定位这些恶意代码，并能够在不影响 BIOS 系统正常使用功能的前提下，清除这些恶意代码。

安全漏洞的存在，通常是与软件或系统的版本、补丁状态相关的。对安全漏洞的检测，需要针对不同的安全漏洞，提取该漏洞存在的特征，包括漏洞代码的特征、漏洞存在的环境特征(如系统或软件的版本号、补丁状态、配置参数等)，建立漏洞库，通过特征匹配的方法实现对漏洞的检测。BIOS 系统存在于 flash 芯片中，BIOS 漏洞所有特征的提取，无论是代码特征还是环境特征，只能通过对 flash 芯片的二进制映像文件的分析得到。

BIOS 中恶意代码的存在则有多种情形，包括对原始 BIOS 代码的篡改、插入新代码、增加新的非标准 BIOS 模块等。同一个厂家的 BIOS 产品，由于 BIOS 版本号不同、编译日期不同、针对的计算机厂商品牌和使用的主板不同，因此 BIOS 映像文件和包括的模块也不完全相同。即使是版本号完全相同的 BIOS 产品，由于编译日期不同，主板厂商也可能根据平台硬件芯片组的变化作了代码上的移植修改，而计算机厂商为了增加品牌机的价值，也会对 BIOS 做客户化，在 BIOS 中增加新功能服务。通常只有同一厂家的同一型号、同一批次的计算机，其 BIOS 产品的 flash 映像文件才有可能完全相同。这些在不同程度上增加了在 BIOS 中检测恶意代码的难度，特别是使得通过采用完整性比较来检测恶意代码的方法基本不可行。图 5.1 显示了计算机 BIOS 产品代码的演化过程。

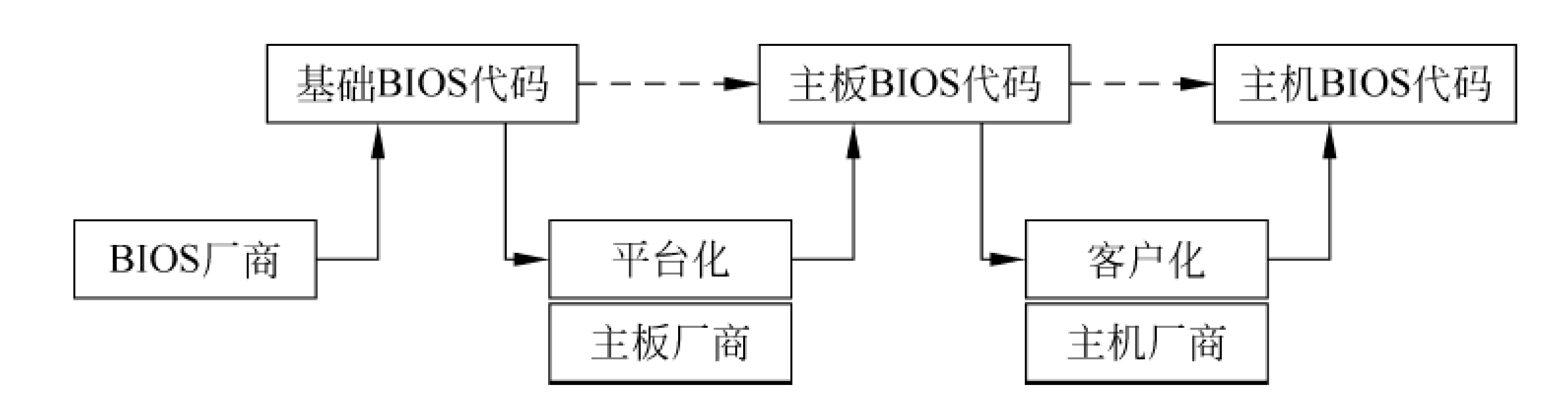


图 5.1　计算机 BIOS 产品代码演化过程

5.2　BIOS 安全漏洞库

针对计算机和网络系统有两种可用的安全漏洞自动检测方法：基于漏洞库的漏洞扫描，以及基于缺陷注入的漏洞测试。前一种方法要求在发现漏洞后，能够提取漏洞的代码特征或者环境特征，将所有被发现的漏洞及其特征存储成漏洞库，通过特征匹配的方法实现漏洞自动扫描[62,63,64,65,66]。基于缺陷注入的漏洞测试则是通过典型的边界输入值或者是模拟攻击，发现漏洞的存在。基于缺陷注入的漏洞测试是一种动态检测方法，通常需要在系统或软件运行过程中才能实施。基于漏洞库的漏洞扫描则是一种静态检测方法，对于存储在 flash 芯片中，运行于操作系统之前的底层固件 BIOS 系统来说，基于漏洞库的漏洞扫描可实施性更好。

5.2.1　固件 BIOS 安全漏洞的特征提取

BIOS 安全漏洞扫描的实现依赖于安全漏洞库。漏洞库决定着检测结果的完备性和准确性。完备性依赖于漏洞库中漏洞的数量和质量，而准确性则依赖于漏洞库中的漏洞特征是否有效，是否能够唯一确定和识别漏洞，而不会出现大量的错报和误报。

通过收集不同 BIOS 厂商（主要是 Phoenix 和 Award 的 BIOS）、不同计算机的 BIOS 产品样本，对 BIOS 二进制映像文件进行模块分解，解压缩，利用反汇编技术（主

要采用 IDA Pro 工具)进行漏洞特征分析、提取、验证,最终获取已发现的 BIOS 安全漏洞的特征。漏洞特征主要包含两种形式:特征字符串和特征二进制码。特征字符串通常包括同漏洞存在相关联的字符串信息,如 BIOS 的版本号、BIOS ID、BIOS 厂商信息、表格头部特征信息、提示和界面信息等。特征二进制码则是通过反汇编从模块代码中提取的一处或多处最能代表漏洞存在的二进制指令码序列。在 BIOS 漏洞库中,漏洞的特征往往是以多个特征的“与”组合构成,称为漏洞的特征组。漏洞特征提取涉及的 BIOS 映像文件结构分解工作在本章稍后描述。

5.2.2 固件 BIOS 安全漏洞的表示

MITRE 安全组织定义的 CVE 要求对漏洞和脆弱点的描述必须满足以下几点[65,66]:列举并能够区别所有已知的漏洞;给每个漏洞分配一个标准的、唯一的名字;在漏洞的多种描述方式中保持独立性;保持开放和共享。CVE 格式的每个漏洞条目由漏洞标识符、状态、引用、内容简述四部分组成,其中漏洞标识符是漏洞的唯一标识。

本书对 BIOS 漏洞的表示方法,满足 CVE 对漏洞描述要求,根据 BIOS 映像文件组织结构的特点(如模块化封装、压缩存储等),为使对 BIOS 系统的漏洞扫描在实现上更具效率,更准确地定位漏洞,采用 6 元组描述所发现的 BIOS 安全漏洞,建立 BIOS 安全漏洞库。

一个 BIOS 安全漏洞 V 其 6 元组表示为:

```
V={
Type,       漏洞类型;
Name,       漏洞名称;
Keys,       漏洞特征组;
Modules,    漏洞关联模块;
Comments,   漏洞描述;
Propose,    漏洞建议;
};
```

Keys＝{keyi} i＝0，1，2，…；

Modules＝{modulei} i＝0，1，2，…；

按照可能造成的危害程度对漏洞进行分类标记；漏洞特征组是1个或多个漏洞特征构成的一维向量；漏洞关联模块则表明，在漏洞特征的提取过程中通过分析得到的该漏洞所涉及的那些BIOS模块，以便在检测过程中能够实现漏洞的快速扫描，准确定位漏洞所处的代码位置；漏洞描述包括该漏洞的危害以及漏洞利用途径；漏洞建议包括对漏洞修补的方案，如打补丁、修改配置、升级BIOS系统等。

利用所建立的BIOS安全漏洞库可以对BIOS实施安全漏洞扫描，本章在这方面所作的尚是一些基础性的研究工作，更多BIOS安全漏洞的发现和整理，则需随着BIOS安全研究的深入和扩展进一步充实，以求能够更全面、更准确地对计算机BIOS系统进行漏洞扫描。

5.3　基于语言的固件恶意代码检测

基于语言的安全验证一直是安全界一个重要的研究领域，同完整性验证维持信任关系的方法相比较，语言验证可用于动态建立安全信任关系。基于语言的安全验证是指，当从一个非信任源获取并装载可执行代码时，在代码执行前，要求同时装载该可执行代码附带的一个安全证书，通过对证书的验证以证明代码的执行将会是安全的。可执行代码的安全证书由代码提供者在编译代码时通过编译器自动生成。

5.3.1　语言验证的安全原理

高级编程语言的编译器在对代码编译的过程中，通过对代码的分析，能够获取大量关于代码的信息，如类型信息、变量值的限制条件信息、代码结构性信息、命名信息

等。这些信息可用于检查代码可能存在的类型错误,并可应用于对代码的进一步优化。编译完成后,传统的编译器往往会丢弃这些额外的辅助信息,只留下编译得到的指令序列,不保留任何结构性特征和其他可识别的源代码特征。

从语言验证的角度来分析,这些额外的信息包含了编译形成的目标代码执行安全性的有效辨别信息。如果代码的装载者能够获知这些只有编译器才能获知的额外信息,则可以在被装载的代码执行前,从一定程度上辨别验证该代码的执行过程是否安全,代码的执行是否存在违反安全政策的行为。

基于语言的安全验证要求编译器在编译源代码的过程中,将这些额外信息按照一定的要求同目标代码一起保留下来,并且作为可执行目标代码的一个结构成分同目标代码一起提交给代码装载者。这部分同目标代码一起封装的额外信息就称为该目标代码的安全证书。当目标代码被装载时,安全证书也同时被装载。装载者使用一个称为语言安全验证器的部件,首先对目标代码及其安全证书进行检查,判断其代码中是否存在可能违反安全政策的部分。如果验证成功通过,则允许代码执行,否则禁止代码的执行。

图 5.2 揭示了语言安全验证的过程。其中编译生成目标代码和安全证书的过程属于开发过程,同代码最终的安全验证和执行过程是分离的,不属于信任建立过程的范畴。

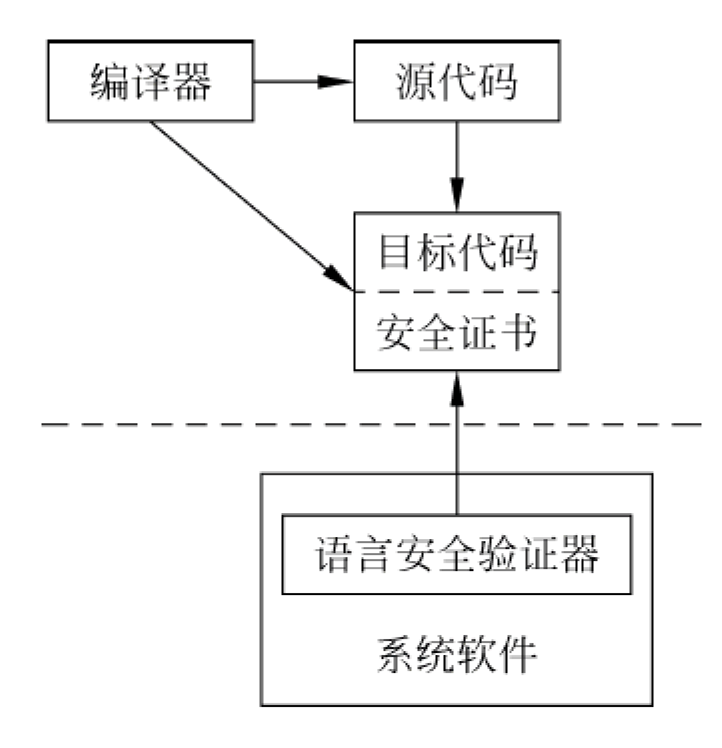


图 5.2　基于语言的安全验证

5.3.2　典型语言验证系统

基于语言的安全验证，首先要求设计一种安全的编程语言。Java 可能是目前能够达到大范围实用的第一种支持语言安全验证的高级编程语言[67]。Java 设计者最初是为了采用这种技术来防止浏览器中运行的恶意插件。在 Java 的运行环境中包含一个 Java 中间代码 Bytecode 的安全验证器，支持对内存安全、控制流安全以及基于类型系统的语言安全的支持。Java 最典型的安全政策是限制对磁盘输入输出的访问，这是由 Java 设计的应用范畴所决定的。如果 Bytecode 验证通过，才允许代码由 Java 虚拟机解释执行，或者进一步编译成本机机器代码。由于最初 Java 缺乏合适的语义模型，导致 Java 的基于语言的安全验证存在较多的缺陷[18]，随之类型系统被引入到 Java 语言中以解决这些安全缺陷[68,69]。Java 这种基于语言的安全机制，很好地解决了互联网上 Web 应用中存在的对本地资源访问限制的安全需求。

PCC(Proof Carrying Code)[21,70,71,72]是一种针对 C 语言并能作优化的基于注解和形式化逻辑证明的语言安全验证系统。代码提供者首先利用编译器 Touchstong compiler 从源代码生成带有注解的目标代码；然后利用一个自动工具由形式化的系统安全政策和带注解的目标代码生成目标代码的安全逻辑推理，该推理用来证明目标代码是满足形式化安全政策的，目标代码和安全逻辑推理被封装到一起提供给代码使用者；最后，代码使用者在执行代码前，首先执行一个证明检查器来检查目标代码逻辑推理的正确性，判定该代码是否能安全执行。PCC 系统尽管能满足复杂高级语言(如 C 语言)的安全验证和优化需求，但是其产生的目标代码包含的逻辑推理安全证书尺寸往往是目标代码的 3～7 倍，并不适合于在固件系统中应用。

TAL(Typed Assembly Language)[20,73,74]是一种基于类型系统的安全汇编语言，可以看做是 PCC 简化的汇编版本。由于简化成了只针对类型系统进行安全验证，TAL 的安全证书尺寸要比 PCC 的小得多。

无论是 PCC 还是 TAL，在代码验证执行的效率上都要受到较大的影响，目前尚不能实际应用。

5.3.3 基于 ECC 的 OPEN Firmware 恶意代码检测

ECC(Efficient Code Certification)[23]针对 PCC 和 TAL 的缺陷进行改进，只提供有限固定的相对基础的安全保护，但其实现更简单，执行效率更高，而且其产生的安全证书的尺寸只有原始目标代码的 6%～25%，被应用于对 Open Firmware 固件系统进行恶意代码防范[24,25]。ECC 提供的有限基础安全属性包括：

控制流安全：将要执行的下一条指令只能存在于该段代码自身的代码段中，而不会跳转到地址空间中其他随机地址处取指令；

内存安全：程序代码只能访问明确分配给该段代码的数据段内存空间和系统堆空间或者合法的堆栈帧，而不允许随机访问内存空间的其他地址；

堆栈安全：子程序或函数的调用要保持堆栈的平衡状态。

控制流安全、内存安全和堆栈安全三种安全条件必须同时满足，才能证明代码的执行是安全的，否则代码将被归类于恶意代码。

ECC 编译器在生成目标代码注解时，将目标代码标记为块代码(Block Code)和残余代码(Residual Code)。例如，程序块(program block)包括主程序的所有代码，调用块(call block)包括函数体的代码，而表示式块(eval block)包含计算表达式所生成的所有代码。在块代码中，删除所有子块代码后剩下的代码称为该块的残余代码。

ECC 要求控制流满足：

(CF1) 只能通过 jump 指令跳转到程序块(program block)第一条指令开始执行程序块，只能通过 halt 指令结束程序块的执行；

(CF2) 不能通过 jump 或顺序执行指令开始调用块(call block)指令的执行，只能通过 halt 指令或 return 指令退出调用块的执行；

(CF3) 除了程序块和调用块，其他类型的代码块的执行必须通过 jump 或顺序执行指令从代码块的第一条指令开始执行，代码块执行的结束要么通过 jump 指令跳转，要么顺序执行紧随代码块的下一条指令。

定义 5-1　如果一次内存访问操作(读或写)访问的是预先已分配的堆存储单元,或者是堆栈的有效单元,或者是程序常量数据单元,则称这次内存访问操作是安全的。

定义 5-2　如果代码块的执行满足下面特性,称该代码块的执行是安全的:

(**S1**) 代码块执行的退出满足(CF1)～(CF3);

(**S2**) call 指令总是转向调用块的第一条指令;

(**S3**) 进入代码块前和退出代码块后,指针 s(堆栈指针)和 e(环境指针)总是能保持自身值的一致性;

(**S4**) 代码执行退出后,环境值和堆栈状态总是正确的;

(**S5**) 代码块执行中的所有内存访问都是安全的。

假设代码块的所有子块执行都是安全的,则可以引出下面的定义。

定义 5-3　如果代码块满足下面特性,称该代码块是残余安全的:

(**LS1**) 代码块的残余代码保障在进入代码块中的任何子块执行前,子块执行的环境条件和参数条件总是能够得到满足;

(**LS2**) 当残余代码中出现 call 指令时,call 转向的指令总是被调用的子块的第一条指令,并且调用子块的参数条件得到满足;

(**LS3**) 除子块调用外,残余代码总是能够维护指针 s、指针 e 的正确值和堆栈的平衡状态;

(**LS4**) 残余代码总是能够保障在退出代码块后,指针 s 和指针 e 以及堆栈总是处于正确状态;

(**LS5**) 残余代码中所有的内存访问总是安全的。

ECC 安全定理　如果所有代码块是残余安全的,那么所有代码块的执行都是安全的。如果最外层的程序块总是从程序块的第一条指令开始执行,并且执行前环境指针 s 和堆栈总是处于正确状态,则程序块的执行是安全的。

Dexter Kozen 给出了 ECC 安全定理的非形式化证明。ECC 安全定理对安全的定义,局限于一种有限的基础安全策略,即代码执行的控制流安全、堆栈安全和内存安全。而对于拒绝服务性的恶意攻击、关键资源的访问保护问题则无能为力,其所能

检测的恶意代码的类型也非常有限。

另外，采用 ECC 的方法检测恶意代码，要求固件系统的生产过程采用特定的能够生成语言安全证书的编译器，且目前 x86 的指令系统也完全达不到 ECC 对指令系统设计的要求。因此，本书认为采用 ECC 检测和防范恶意代码对于市场上目前占主流的传统公用 BIOS 产品来讲(如 Phoenix BIOS、Award BIOS、AMI BIOS)暂时尚不可行，但语言验证对代码行为安全的预先判定确是很有前景的研究方向。

5.4 基于二进制结构签名的恶意代码检测

由于受到不同硬件平台、主板厂商、BIOS 厂商修补的影响，即使是同一个 BIOS 模块，对不同计算机主板的 BIOS 产品，其模块代码也总是存在差异；即使是同一款计算机，由于批次的不同，其相同 BIOS 模块代码也可能存在差异。因此，通过完整性验证来检测恶意代码对 BIOS 模块代码篡改的方法在这里具有很大的局限性。

二进制代码比较技术[75,76,77]通过对同一代码不同版本的比较，可以获取不同代码版本中发生改变的细节。这种技术在安全领域的补丁分析中有着十分重要的作用，在病毒软件的变种分析中也有广泛的应用。本节借鉴二进制代码结构化比较技术，通过定义合适的代码结构化特征签名，实现将该技术应用到对 BIOS 产品的恶意代码检测问题中。

5.4.1 二进制代码结构化图描述

二进制代码十分晦涩难懂，直接按照字节进行比较匹配是毫无意义的。将二进制代码利用反汇编技术进行预处理(如采用 IDA 工具)，就可以将晦涩难懂的二进制变成汇编指令的代码序列，并采用图形结构来描述二进制代码。本文“图”的概念，是指由“顶点”和“弧”构成的有向图，形式化定义为：

$$Graph=(V,R)$$
$$V=\{x \mid x\in dataobject\}$$
$$R=\{VR\}$$
$$VR=\{<x,y> \mid P(x,y)\wedge(x,y\in V)\}$$

V 是图中数据元素 x 的集合，x 称为图的顶点；VR 是顶点之间关系的集合，若 $<x,y>\in VR$，则 $<x,y>$ 表示从顶点 x 到 y 的一条弧；图的谓词 P(x，y)表示从顶点 x 到 y 的一条单向通路。顶点的入度是指射向顶点的弧的数目，定点的出度是指从顶点射出的弧的数目。

采用两种类型的图来描述二进制代码的结构：一是函数调用图，描述二进制代码中所包含的不同函数之间的调用关系；二是控制流图，具体描述一个函数内部的流程结构。对于非函数/过程包装形式的二进制代码模块，将该模块等同定义为一个函数，采用函数调用图来描述其结构。

函数调用图：描述二进制模块代码中函数之间调用关系的有向图，图中顶点是二进制代码中包含的所有函数，图中的弧代表从一个函数到另一个函数的调用关系。

控制流图：一个函数所有的语句序列可以被划分为若干基本块。基本块是一个连续语句序列，控制流从它的开始进入，从它的末尾离开，不允许有中断或分支(末尾除外)。控制流图的顶点由基本块组成，表示计算的指令序列，顶点之间的弧表示流向。每一个函数的控制流图只能有一个入口(入度为 0 的点)，但可能存在多个出口(出度为 0 的点)。图 5.3 给出了实际分析中构造的某函数控制流图的示例。图中只有唯一的入口顶点 V0 和出口顶点 V9。

对于二进制代码，首先通过反汇编识别出代码中的函数，绘制出函数之间的调用关系图，然后为每一个函数划分基本块，绘制出每个函数的控制流图。这样，通过把二进制代码抽象成一张函数调用图和若干张函数控制流图的描述，将二进制代码比较问题转化成在图中寻找顶点之间对应关系的问题。对应关系包含了函数之间的对应和基本块之间的对应，从而清晰地显示出不同版本、不同时期内 BIOS 产品中模块代码的变化。

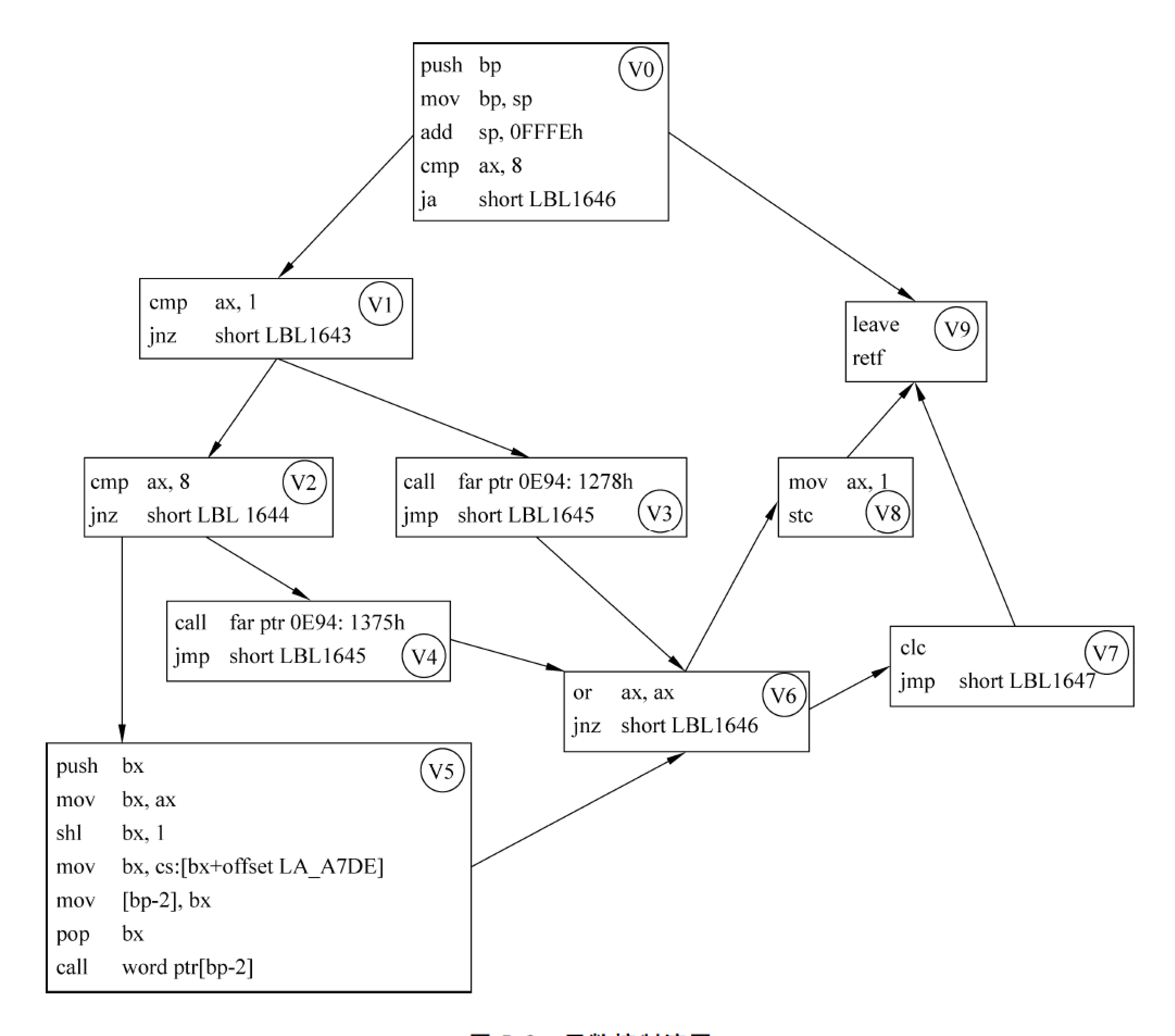


图 5.3　函数控制流图

5.4.2　二进制函数结构化特征签名

二进制代码的比较需要在函数和基本块两个层面进行。在函数层面比较二进制代码，就是要在两份不同版本的函数调用图之间进行顶点之间的映射。同一个功能模块的二进制代码发生改变存在多种情况：由于编译器类型或版本的不同造成二进制代码的改变通常是基本块内部的语句序列和寄存器分配差异；而代码补丁或非法

嵌入的恶意代码往往会改变函数局部的控制流程或执行分支。为了适应这种情况，二进制代码比较过程中函数的映射并非总是函数代码完全匹配的映射，而更多的时候是一种模糊映射。函数的映射基于函数的特征签名进行，而这种函数的特征签名需要适用于模糊映射的需要。

函数控制流图准确地描述了函数内部代码的结构。控制流图中基本块和弧的数目一定程度反映了函数的规模和流程。由于函数控制流图是通过反汇编技术获取的，在对基本块进行特征提取和比较前，不可能实现对控制流图中的顶点进行对应，这给函数比较映射造成了一定的困难。

定义 5-4　顶点特征　控制流图中的顶点特征 Sig(v)=(ID(v),OD(v))，其中 ID(v)代表顶点 v 的入度，OD(v)代表顶点 v 的出度。

为了方便描述和比较，规定两个顶点特征之间的比较关系为：

Sig(vi)= Sig(vj)当且仅当 ID(vi)=ID(vj)并且 OD(vi)= OD(vj)；

Sig(vi)> Sig(vj)当且仅当 ID(vi)>ID(vj)或者 ID(vi)=ID(vj)并且 OD(vi)>OD(vj)；

Sig(vi)< Sig(vj)当且仅当 ID(vi)<ID(vj)或者 ID(vi)=ID(vj)并且 OD(vi)<OD(vj)；

定义 5-5　控制流图的结构特征　具有 n 个顶点的控制流图的结构特征 Sig(g)是图中所有顶点 vi 的顶点特征 Sig(vi)的非递减序列，其中，$0\leqslant i\leqslant n-1$。

顶点的出度和入度能较好地反映图中弧的数目和流向。在两个图中顶点和弧不能确定对应的情况下，控制流图的结构特征 Sig(g)能够一定程度地表示图中顶点之间的关系。

定义 5-6　函数的特征签名　函数的特征签名 Sig(f)=(α,β,Sig(g))，其中 α、β 分别代表函数 f 的控制流图中的顶点的总数和弧的总数，Sig(g)代表函数 f 的控制流图的结构特征。

函数特征签名中的 α 和 β 表达了控制流图的规模，而函数的结构特征 Sig(g)描述了控制流图的结构，三者结合能够很好地标示和识别函数。

在对函数进行映射后，需要进一步对函数中的基本块进行映射，即识别和标定

函数控制流图的顶点，建立基本块顶点间的映射，以便更进一步发现和确定函数中代码的变化。对基本块的特征选取必须兼容编译器对于二进制代码的优化和寄存器分配差别造成的不良影响，即可能造成的基本块语义相同，而指令次序和操作数寻址方式发生了变化。如果不能兼容基本块在这方面的变化，则容易产生较多的误报，把语义完全相同的基本块报告为不同。对于这类问题，可以利用小素数乘积的分解唯一性性质和乘法运算的可交换性质，采用基本块的小素数乘积来作为基本块的特征[77]。

设 A＝{a1，a2，…，am}表示汇编指令的集合，共有 m 条不同的指令助记符，Pm＝{3，5，…，ρm}为前 m 个素数的集合。建立一一映射：

$$\tau: A \rightarrow Pm, \tau(ai) = \rho i$$

该映射为 A 中的每一条指令助记符分配一个唯一的小素数。

定义 5-7　基本块的特征签名　基本块的特征签名 $Sig(b) = \prod_{i=1}^{n} \tau(ai)$，即为基本块中包含的所有 n 条指令所对应的小素数乘积。

如果两个基本块对应的小素数乘积相同，则说明两个块中指令序列所包含的指令助记符集合是相同的，而指令次序，则有可能完全相同，也有可能不同。而且，在指令助记符集合相同的情况下，允许具体指令采用不同的寄存器分配和操作数寻址方式，从而很好地解决了编译器可能带来的相同语义基本块之间的指令差异。当基本块的二进制代码的小素数乘积相同时，可以认为这两个基本块是语义相同的基本块。

5.4.3　基于结构化特征签名的恶意代码检测过程

基于图的二进制函数特征签名和基本块的特征签名从两个不同的层次对二进制函数代码进行了描述：函数特征签名用于从函数调用关系图中识别出相同或相近的二进制函数代码，而基本块特征签名则可以用于对函数中代码的进一步比较，识别函数代码语义上的变化。

基于图的二进制结构描述和特征签名可以用于对二进制代码中可能包含的恶意代码和其他可疑代码进行检测。检测前需要广泛收集恶意代码样本和典型的 BIOS 模块样本。

对于已知的拥有二进制代码样本的恶意代码，采用基于图的二进制结构描述提取恶意代码主函数的特征签名或恶意代码最具代表性的基本块的特征签名，这些特征签名连同恶意代码一起构成 BIOS 安全检测的恶意代码样本库。

对于典型的 BIOS 模块样本，假设样本收集的来源是可信的，并经过代码分析确认模块样本中所有代码的功能都是已知并无害的，即典型 BIOS 模块样本是安全可信的。对于这些典型的模块样本，同样采用基于图的二进制结构描述生成模块代码的函数调用关系图和函数的控制流图，提取各个函数的特征签名和基本块的特征签名。这些典型 BIOS 模块样本同对应生成的函数调用关系图和特征签名一起，构成 BIOS 安全检测的 BIOS 模块样本库。BIOS 模块样本库需要按照不同厂家、不同版本的 BIOS 产品对模块样本进行分类存储。

BIOS 安全检测过程中，在完成对被检测计算机 BIOS 产品的采样、BIOS 映像文件模块分解、解压缩，得到独立的 BIOS 代码模块过程后，对恶意代码检测的过程为：

(1) 对所获得的所有 BIOS 代码模块，采用基于图的结构描述生成模块的函数调用图和控制流图，提取函数的特征签名和基本块的特征签名。

(2) 根据恶意代码样本库中恶意代码的特征签名在 BIOS 代码模块中搜索匹配恶意代码，即将恶意代码的主函数根据特征签名在 BIOS 代码模块中寻找能够匹配特征签名的配对函数。

(3) 将被检测产品的模块样本与 BIOS 模块样本库中的二进制代码按照模块、函数、基本块的顺序依次进行搜索配对。模块配对的依据是模块类型和函数调用关系图，函数和基本块配对的依据是各自的特征签名。完全配对的函数和基本块表明其中不存在可疑代码，不能配对的函数，需要进一步进行模糊匹配，找到最接近的函数，再进一步通过基本块的比较找出函数中的代码变化。这种未知语义的代码变化称为可疑代码。对可疑代码的分析需要人工进行反编译和阅读分析，识别变化部分的代码语义功能，以确认可疑代码是良性代码还是恶意代码。

5.5 BIOS 产品结构分析

BIOS 在 flash 芯片中的存储是按照类似于一个简单的文件系统来构成的，其构成的最小成分是模块。不同厂商 BIOS 产品 flash 映像文件的结构都不一样。而且，为降低 BIOS 产品的成本，减少 BIOS 映像文件需要占用的存储空间，BIOS 映像文件中的许多模块都是经过压缩后再存储的。因此，对 BIOS 进行漏洞扫描和恶意代码检测，都需要首先将 BIOS 的 flash 映像文件解析成模块，对压缩的模块再进行解压缩，才能得到真实的数据或代码。

由于 BIOS 厂商对 BIOS 产品的技术垄断和非开放性，导致 BIOS 映像文件的结构各不相同并且可参考的开放性资料很少，所以对 BIOS 产品结构分析大部分需要通过逆向工程的方法来获取。本书以在国内市场占主流的 PC BIOS 产品 Award BIOS 和笔记本 BIOS 产品 Phoenix BIOS 为例分析了典型的映像文件结构和模块结构。

映像文件对应 4GB 内存的最高地址的 8KB(某些产品为 64KB)称为 Boot Block，该部分代码不压缩，包含计算机开机立即执行的指令(在映像文件的最后 16 个字节起始处，对应 4GB 内存的 0FFFFFFF0H 地址)，以及其他一些平台初始化的基础代码。

5.5.1 Award BIOS 映像文件和模块结构

传统的 Award BIOS 映像文件采用模块的线性排列结构，最高端对应的是 Boot Block，然后依次顺序排列存放其他模块。模块与模块之间允许存在自由空间(被填充为 0FFh)。模块的内容由模块头部结构和模块的数据/代码构成。典型的模块头结构为：

```
Head={
5Byte       "-lh5-"
1Byte       头部长度(以字节为单位);
1Byte       头部 CRC 校验和;
1Byte       压缩算法标识;
2 Word      压缩后模块长度(以字节为单位);
2 Word      原始模块长度(以字节为单位);
2 Word      模块类型标识;
1 Byte      ?;
1 Byte      ?;
1 Byte      模块对应文件名长度;
128 Byte    模块对应文件名
}
```

Award BIOS 通常采用 LZHUF 压缩算法对模块进行压缩。而模块类型标识指明了模块包含的数据或代码的类型和作用,如 0x40000000 表示模块是 BIOS Logo 图片,0x40010000 表示一个 CPU 微代码模块等。

5.5.2　Phoenix BIOS 映像文件和模块结构

传统 Phoenix BIOS 的映像文件采用模块的单链表存储结构,最高端的 Boot Block 不包含在单链表中。各个模块通过模块头部中的地址指针单向连接在一起,形成一个模块链表。模块头部结构如下:

```
Head={
2 word      指向前一个模块的地址指针;
3 byte      通常为"011";
1 byte      同类型模块序号;
1 byte      模块类型;
1 byte      模块头部长度(以字节为单位);
1 byte      压缩算法;
2 word      ?;
2 word      原始模块长度(以字节为单位);
2 word      压缩后长度(以字节为单位);
```

```
2 word      ?;
2 word      ?;
}
```

在传统 Phoenix BIOS 的产品中,采用单个英文字符表示模块类型,如 A 表示 ACPI 模块,E 表示 BIOS setup 模块,G 表示用于对压缩模块执行解压缩操作的模块等。若存在多个同一类型的模块,则以序号顺序表示。

5.6 BIOS 安全检测模型

BIOS 安全检测模型表明了 BIOS 安全检测的方法和内容。本节建立了一个 BIOS 安全检测的自反馈闭路模型用以指导 BIOS 安全检测系统的开发。BIOS 安全检测模型如图 5.4 所示。

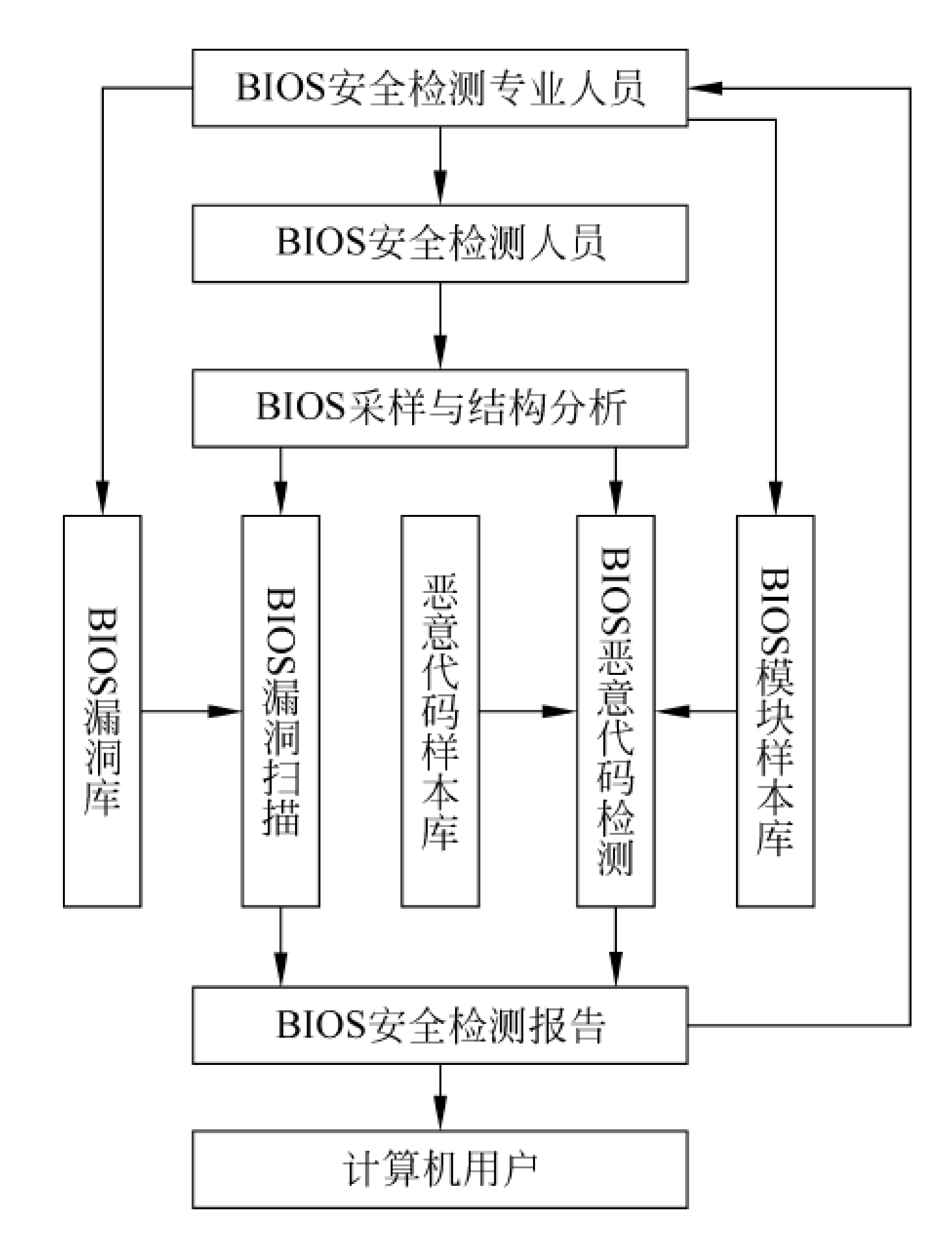


图 5.4 BIOS 安全检测模型

该模型中，BIOS漏洞库、BIOS模块样本库、恶意代码样本库根据BIOS安全检测专业人员预先研究分析成果建立。BIOS漏洞库保存已发现的BIOS漏洞的描述，其漏洞特征是漏洞扫描的依据。BIOS模块样本库根据不同BIOS产品进行分类，建立已知的BIOS模块的二进制代码结构签名，并且这些模块是已通过安全分析确认不包含恶意代码的安全模块。恶意代码样本库保存有已知恶意代码的特征描述和二进制结构签名。

BIOS安全检测人员经过专业人员的培训指导，采用检测系统对被检测计算机BIOS进行采样并作安全检测，确认被检测计算机BIOS系统是否存在安全漏洞，是否被植入恶意代码。检测所生成的报告反馈给计算机用户，同时也反馈给BIOS安全检测专业人员。

若检测结果发现未知的BIOS功能模块或未知的可疑代码，安全专业人员进一步分析该模块或代码的功能，并对其提取特征值和二进制结构签名，根据分析数据和结果充实BIOS模块样本库和恶意代码样本库。安全专业人员在研究分析检测的过程中，根据研究结果持续对BIOS漏洞库进行更新升级。

BIOS安全检测结果的反馈，补充和完善了BIOS安全漏洞库、BIOS模块样本库以及恶意代码样本库；反过来，这些库数据的不断补充和完善又进一步扩大了BIOS安全检测结果的覆盖范围，提高了检测的准确性。

5.7 BIOS安全检测系统的实现

通过对计算机BIOS固件的安全检测及安全修补，能够有效地减少、降低、消除计算机BIOS存在的安全漏洞，防止计算机BIOS中代码被恶意修改或被植入木马程序，达到增强计算机信息系统安全的目的，为计算环境和网络环境的信息安全、信息保密提供牢固可靠的底层防范和保护能力。

5.7.1 BIOS 安全检测的内容和流程

BIOS 的安全检测是一种自动实施的静态检测，其检测的目的物是从 BIOS flash 芯片中提取出的 BIOS 二进制映像文件，检测的内容包括漏洞扫描和恶意代码检查。

BIOS 安全检测的流程如图 5.5 所示。

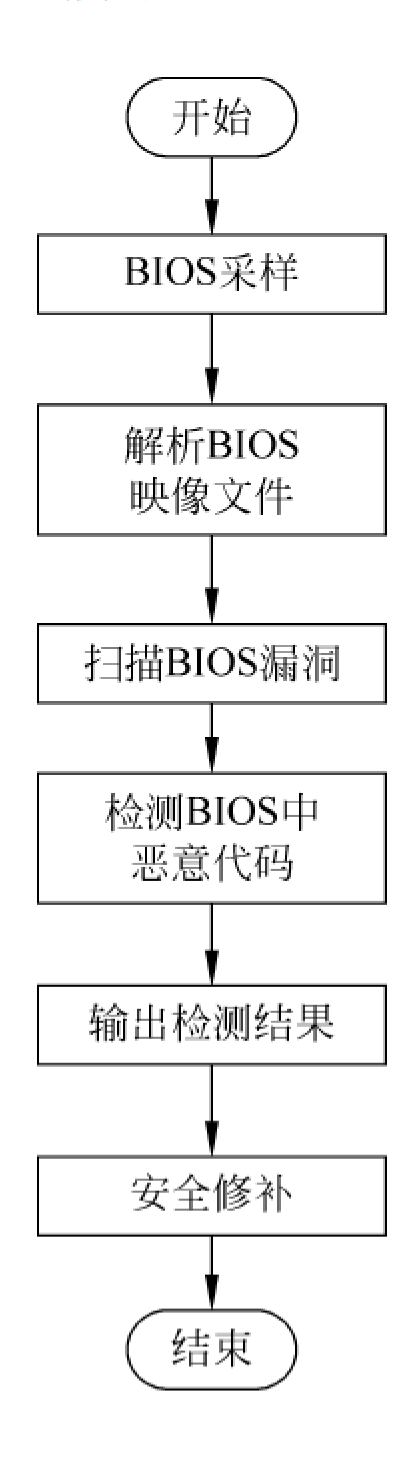


图 5.5 BIOS 安全检测流程

为了不破坏计算机主板，以软件读取的方式对计算机主板上的 flash 芯片中 BIOS 进行采样，形成 BIOS 映像文件。

对 BIOS 映像文件的解析过程，就是从 BIOS 中提取 BIOS 基本信息和结构信息的过程。基本信息包括 BIOS 厂商信息、序列号(BIOS ID)、产品发布时间、版本号、版权信息等。BIOS 的结构信息包括 BIOS 的模块组成、数量、类型和尺寸等，将每个

模块分解结果存储到独立的临时文件中，以备进一步漏洞扫描和恶意代码检测使用。对于被压缩存储的模块，分解时要进行解压缩。不同 BIOS 产品中包含的模块数量可能有较大的差别，例如，联想开天系列计算机采用的 Award Modular BIOS v6.0 的产品一般只包含十几个模块，而 IBM T42 系列的笔记本使用的 Phoenix 4.0 Release 6.0 的 BIOS 产品则一般包含 70 个以上的模块。

5.7.2　BIOS 安全检测系统结构

BIOS 安全检测系统结构设计的目标，既要满足对单机 BIOS 进行安全检测的需求，又能通过局域网对网内多台计算机通过网络进行集中检测的需求。所设计实现的 BIOS 安全检测系统结构如图 5.6 所示。

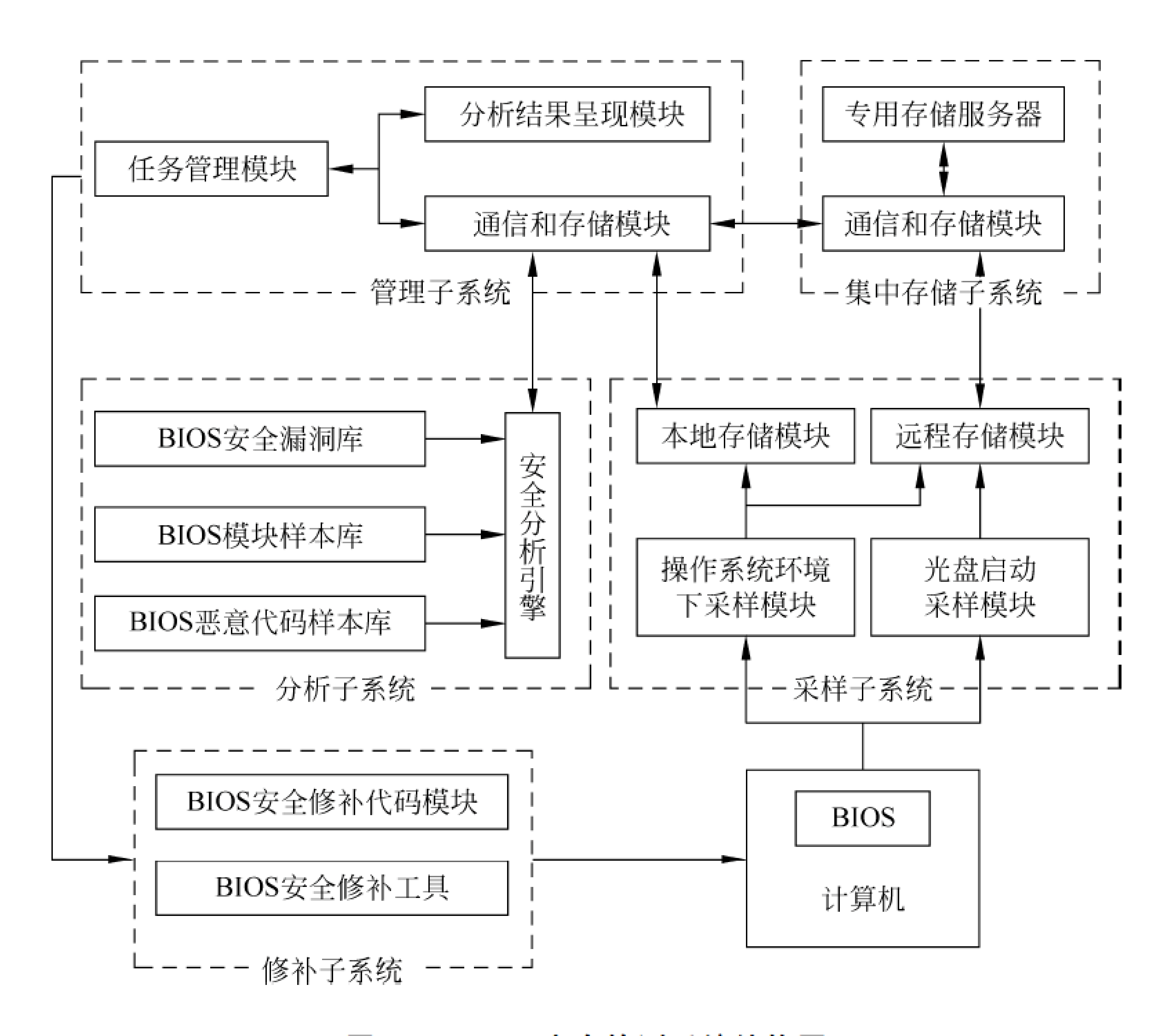


图 5.6　BIOS 安全检测系统结构图

BIOS 安全检测系统包括采样子系统、分析子系统、管理子系统、修补子系统、存储子系统五个部分构成。提供本地单机检测和集中式远程检测两种检测模式。采样子系统完成对计算机主板上存储 BIOS 的 flash 芯片的本地采样或远程采样，读取 flash 芯片内容并存储成二进制映像文件。分析子系统对采样 BIOS 数据进行安全分析检测，生成检测的技术性报告。管理子系统建立和管理安全检测任务，呈现检测结果。修补子系统根据 BIOS 安全检测结果对 BIOS 映像文件实施安全修补，并将修补后的 BIOS 映像文件写回计算机主板上 flash 芯片中。采用集中式远程检测模式时，存储子系统采用 TCP/IP 协议通信接收来自采样子系统的 BIOS 采样数据，提供对 BIOS 采样数据的集中存储和检测结果的存储管理。

对 BIOS 的安全修补采用自动和人工手工修补相结合的方法进行。由于重写主板 BIOS 风险比较高，如果修补不正确或者重写过程出现差错，就会导致计算机不能启动。因此，对 BIOS 的安全修补，通常由 BIOS 专业人士或训练有素的安全检测人员来进行，普通用户要慎用这一操作。

由于计算机品牌和 BIOS 产品厂商、版本的复杂性，为适应不同环境下的 BIOS 采样需求，系统采用两种采样模式：一种是进入 Windows 操作系统后，在操作系统下执行系统提供的采样工具软件进行采样；另一种是使用系统提供的启动光盘，从光盘启动进入 Linux 操作系统，在 Linux 操作系统下运行采用工具软件进行采样，而无须进入被采样计算机硬盘上的主机操作系统。前一种采样方法简单方便，并且可通过网络实时传输采样结果。后一种采样方式具有更广泛的适应性，能够在市场上绝大多数计算机上成功对 BIOS 进行采样。

经过对市场上 BIOS 产品进行调研、收集和分析，本书所实现的 BIOS 安全检测系统能够对目前市场上最为普遍的两类 BIOS 产品（Phoenix BIOS 和 Award BIOS）进行安全检测。该系统可广泛应用于对保密网中的保密计算机的安全检查和安全评估工作中。该系统已经成功地进行了产品转化。

5.8　本章小结

目前计算机中广泛使用的传统BIOS产品存在较大的安全风险：BIOS产品存在安全漏洞，可能被恶意者利用实施对计算机系统的攻击；恶意者也可能通过网络综合利用各种入侵手段，将恶意代码植入到计算机BIOS芯片中。恶意代码能够在BIOS运行阶段获取系统控制权，也能够在操作系统运行后再获取系统的控制权。在BIOS产品来源无法控制的情况下，扫描BIOS产品存在的安全漏洞，检测BIOS芯片中是否被嵌入恶意代码，是保障计算机BIOS安全的一种重要手段。

本章根据第4章BIOS安全漏洞分析结果，对漏洞提取特征，规范漏洞条目的表示，建立了第一个BIOS安全漏洞库。讨论了基于语言的行为验证、二进制结构签名、特征提取和匹配等多种恶意代码检测方法。分析了Phoenix和Award两种BIOS产品的映像文件和模块结构，开发了一种计算机BIOS安全检测工具。该检测工具获得了中共中央办公厅科技进步二等奖，并由北京电子科技学院-中科院高能物理研究所网络安全联合实验室成功地进行了产品成果的转化，集成到苏州中科林华科技有限公司生产制造的“神探I号-终端全方位安全检查系统”产品中。



第 6 章 可信固件开发的安全策略和模型

受 CPU、南北桥芯片设计技术能力不足的限制和传统 BIOS 技术垄断的影响，我国一直缺乏自主知识产权的国产化固件 BIOS 产品。因此，对目前市场上占主流的美国和我国台湾地区的传统 BIOS 产品进行安全漏洞扫描、安全威胁分析和检测，是在现存条件下，对国家信息安全底层系统安全的有力补充和保障。但是，要构造我国完整的信息产业链，实现信息安全各个环节的可控和可信，设计和开发自主知识产权的国产化安全 BIOS 产品就成为必需的环节和必经之路。本章讨论信息安全领域安全和可信概念的关系，明确安全固件开发的目标——可信固件。

6.1 可信与安全的关系

在信息安全领域，可信(Trust)和安全(Security)是两个经常混淆使用的概念，两者关系到底如何，尚没有正式文字对此有全面明确的阐述。本文认为，明确区分可信和安全两个概念，对于信息安全发展目标的明确化和开发手段的确定化是有帮助的。混淆两者的概念，容易对计算机和信息安全发展的方向和技术手段造成误导。

6.1.1　可信与安全概念使用的历史阶段划分

首先我们从发展的眼光来考察在信息安全的不同历史发展阶段对这两个概念的使用情况。

第一阶段：1967—1983 年。

1967 年 10 月，美国国防科学部成立了一支研究小组，开始了对计算机安全保障问题的研究，重点研究计算机系统中的资源共享和远程访问的机密信息保护问题[78,79]，从此拉开了计算机安全研究的序幕。随后在 1972 和 1973 年，美国国防部根据其研究成果，制定发布了统一的国防部计算机系统使用安全要求、管理控制政策和技术手段[80,81]。1972 年，作为承担美国空军的一项计算机安全规划研究任务的研究成果，J. P. Anderson 在其研究报告[82]中提出了引用监控机(Reference monitor)等重要安全思想。而 D. E. Bell 和 L. J. Lapadula 在 1973 到 1976 年间提出并逐步充实完善的 BLP 安全模型[83,84,85]成为第一个可证明的安全模型，也是最有影响力的关注机密性的安全模型之一。在这个阶段，一系列的安全操作系统被设计和开发出来[10]。

这个阶段，使用的几乎都是“安全”这个概念。

第二阶段：1983—1993 年。

1983 年美国国防部颁布了《可信计算机系统评估准则》[79](TCSEC，也称桔皮书)，开始使用“可信”这个概念和词汇。1985 年，美国国防部对 TCSEC 进行了修订[86]。TCSEC 主要针对的是保密性，重点关注的是通用操作系统产品的安全性。在随后的一段时间内，各国相继颁布了各自相关的安全评估准则，其中部分准则沿用了 TCSEC 的“可信”概念，包括：1987 年，美国国防部相继颁布的针对可信计算机数据库、可信计算机网络等的系列评估标准，也称彩虹系列；1993 年，加拿大颁布的《可信计算机产品评估准则》[87](CTCPEC)。而其他一些准则倾向于使用“安全”这一概念，包括：1991 年，英、法、荷、德联合提出的《信息技术安全评估准则》[88](ITSEC)；1993 年，美国公开发表的《信息技术安全联邦准则》[89](FC)。

这一阶段，可信和安全两个概念同时混合使用，但从 TCSEC 对可信计算机的定

义来看，这一时期“可信”和“安全”两个概念是等价的，具有相同的含义。

第三阶段：1993—1999 年。

1993 年，出于对评估准则混乱状况的不满意，CTCPEC、FC、TCSEC、ITSEC 的发起者采取联合行动，准备将各自独立的准则集合成一组单一的、能被广泛使用的信息技术安全准则，并在 1996 年 1 月完成了这一准则的 1.0 版本，即《信息技术安全评估通用准则》v1.0(CC v1.0)，1997 年完成 CC v2.0 的测试版，1999 年 CC v2.1 成为国际标准 ISO/IEC15408：1999[90]。在这一标准中，统一采用了“安全”这个概念和词汇，受该标准的影响，同期安全产业界和学术界也基本更多地采用了“安全”这一概念。

第四阶段：1999—2003 年。

1999 年，IBM、HP、Intel、微软等著名 IT 企业发起成立了可信计算平台联盟(Trusted Computing Platform Alliance，TCPA)，该组织不仅考虑信息的机密性，更强调信息的真实性和完整性，更注重安全的产业化。这一阶段，可信概念被重新提起，但其意义已经发生了改变，由早期与“安全”几乎相同的概念，演化成更注重实体真实性和信息完整性的概念。这一阶段，无论产业界还是学术界对可信计算尚存在许多争论，安全这一概念也同时广泛地使用并且占据主要地位。

第五阶段：2003—2008 年。

2003 年，TCPA 改组成可信计算组织 TCG，推出了一系列可信计算的规范标准，同时可信计算模块 TPM 产品的出现，将可信计算推向了一个新高潮。这一阶段，“可信”概念和词汇的使用远远超过“安全”，在众多媒体和厂商的炒作下，TCG 的可信计算甚至被当成了计算机安全的终结方案和技术手段，似乎可信计算就是安全界黑夜里的一盏明灯，为计算机安全指明了终极道路。

不同历史发展阶段对可信和安全概念的使用可以用图 6.1 来表示。

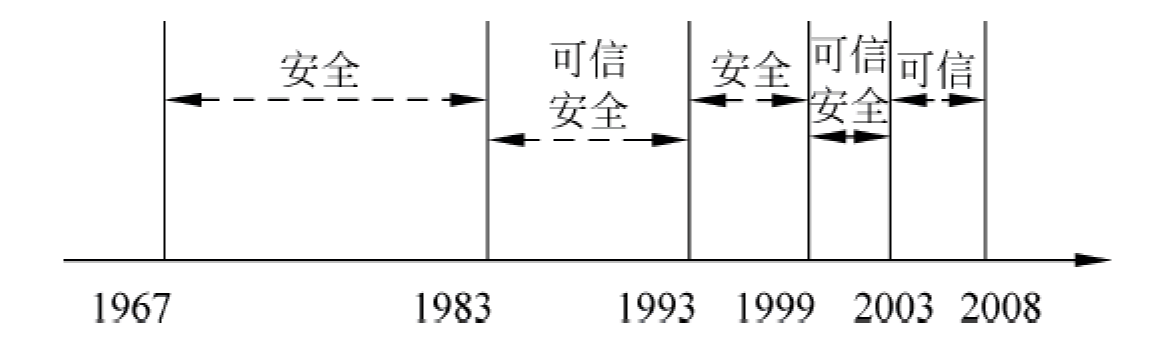


图 6.1 不同历史阶段可信和安全概念使用情况

6.1.2　可信与安全概念的内涵比较

下面考察在不同发展阶段，安全和可信概念的定义和认识需求。

在不同历史阶段，对安全的定义、需求和认识基本是一致的。这其中 ISO/IEC 15408 中对“安全”的阐述具有代表性：

定义 6-1　安全-ISO/IEC 15408：安全就是使被保护的资产免受潜在威胁的滥用，这种滥用包括但不限于对被保护资产的机密性、完整性以及可用性的破坏。

美国国防部发布的 TCSEC[79,86]中对可信计算机系统的定义则是在 1983—1993 这一阶段中对可信定义的典型阐述：

定义 6-2　可信-TCSEC：可信计算机系统就是采用了足够的硬件和软件完整性保护措施以保障系统的可用性，使得系统能够有效处理机密信息。

虽然 TCSEC 对可信计算机系统的定义突出地强调了对系统处理机密信息的重点关注，但同时也表述了对系统完整性和可用性的要求。因此说，在上述第二阶段中使用的可信的概念，实际上同安全概念是等同的。这也是为什么 CC 标准和 ISO/IEC 15408 国际标准在第二阶段的可信和安全概念混合使用后，能够在第三阶段将两者统一规范为一致采用安全概念的主要原因。

TCPA 和 TCG 重新提出的可信概念，由于其可信概念的不明确性，以及可信概念和所采用技术的不一致性，再次引起了可信和安全概念的使用混乱。TCG 在文献[32]中对可信的定义为：

定义 6-3　可信-TCG：如果实体以预期的特定行为完成指定的目标，则实体是可信的。

可见，TCG 定义了一个很模糊的可信概念。从预期安全的角度出发，可以这样理解 TCG 对可信的定义：①实体应该完成的指定目标是明确的，即实体的功能预期是明确的；②实体除了完成预期的指定功能外，不应该完成其他额外的、非预期的功能；③对实体特定行为的预期是安全的，即实体行为不破坏系统或信息的完整性、机密性和可用性。

因此，从预期安全的角度来讲，TCG 定义的可信概念是等效于安全概念的。

遗憾的是，TCG 随后提出的系列规范标准、技术方法都没有贯彻这种可信概念，没有实施对实体行为的验证。在 TCG 的文档中，可信度量等同于对实体的完整性测量(完整性同时关注信息的真实性和正确性)，而我国国家密码管理局 2007 年 12 月颁布的《可信计算密码支撑平台功能与接口规范》[33](TCM 标准，对应于 TCG 的 TPM 标准)，也沿用了 TCG 的这一提法，将可信度量完全等同于完整性度量。

很明显，完整性度量只能保证实体或信息的真实性和正确性，但不能保证实体行为的安全性。因此说，在实际的产业化实施中，TCG 定义的可信已经蜕变成一种实体之间的静态关系，而不是通过强制规范实体执行过程中的行为来保障实体执行的安全。从这一点来讲，TCG 的可信只是解决了部分安全问题，可信的不一定就是安全的。

6.1.3 对信息安全研究的指导作用

安全要求实体的行为以不破坏系统和信息的机密性、完整性和可用性为原则。等效于安全的可信要求不仅对实体的执行结果做出安全预期，而且对实体的行为过程也要求进行验证(行为的可信度量)。静态的可信关系不足以保证实体的行为总是安全的。

软件行为学认为，主体的可信性是主体行为的一种统计特性，而且是指行为的历史记录反映主体行为是否违规、越权以及超过范围等方面的统计特性[91]。软件行为学试图通过软件行为的统计来建立实体之间动态的可信关系，但这种动态关系是通过行为的事后统计建立起来的，甚至需要以承受不安全行为对系统造成损害为代价，这对于很多安全环境和需求来讲，是不可以接受的。

基于语言的安全验证在程序代码执行前对代码行为进行静态检查和预测，以期发现被加载执行的代码中不符合安全策略的行为。设想可信度量既包括 TCG 的完整性度量，以保证实体的真实性和正确性，又包括这种对实体预期行为的度量，以保

证实体行为的安全性，则这种可信度量足以保障可信实体的安全性，达到可信即等效于安全的目标。

正确认识可信和安全的关系，可以帮助我们认识到目前TCG可信计算的局限性，继续深入探讨和研究行之有效的安全理论和技术途径。

6.2 固件在可信计算体系中的地位和作用

可信计算环境要求从硬件层到软件层，组成系统的各个部分都是可信的，任何不能证明自身是可信的部件，如恶意代码、病毒、木马等，都被排斥在这个系统之外。我们称这种计算和网络环境为健康的可信计算生态环境。建立可信计算生态环境，要从加强计算终端的硬件安全、固件安全、操作系统安全等底层安全入手，从源头上控制不安全因素。

TCG的可信计算以可信硬件为基础，通过信任传递在硬件、固件、软件之间构建信任链，任何实体只有通过可信度量才被准许进入系统中。TCG规定可信平台模块(Trusted Platform Module，TPM)是可信计算的硬件安全基础。TPM是一个微控制器芯片，通常集成在计算机主板上，用于存储密钥、数字证书、完整性度量值，并具备数字签名、密钥交换、加密解密等安全运算功能。

TCG的可信平台具有三个核心根，即可信度量根(Root of Trust for Measurement，RTM)、可信存储根(Root of Trust for Storage，RTS)和可信报告根(Root of Trust for Reporting，RTM)。RTM是完整性度量的计算单元，是信任链建立的起点，负责实施RTM的代码通常称为CRTM(Core Root of Trust for Measurement)。RTS是能够保护存储完整性摘要值并保留其计算顺序的TPM计算和存储单元。RTR则是指能够可靠地向外部平台或实体报告RTS存储的信息的TPM计算单元。

CRTM(Core Root of Trust for Measurement)是可信度量的根执行代码，也是信任链中最早被执行的指令代码。作为可信计算的硬件基础，将CRTM包含在TPM

中当然是最合适、最安全的，如图 6.2 所示。这样，可以利用 TPM 硬件的密封存储和密封计算特性来保护可信计算的三个核心根。

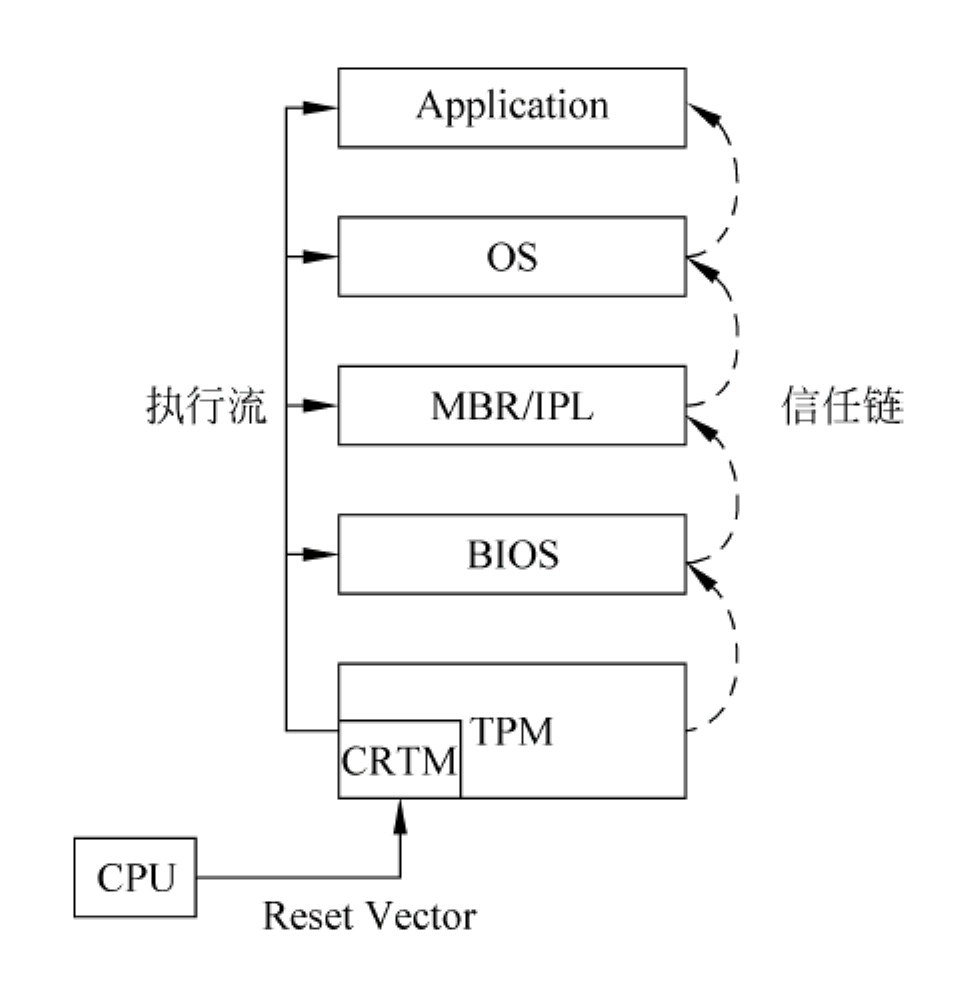


图 6.2 CRTM 包含在 TPM 中的信任链

将 CRTM 包含在 TPM 芯片中是一种比较理想的方案，但是在现有的计算机体系架构下，这一方案的实现并不可行。现有的计算机体系结构，CPU 上电后拾取并执行的第一条指令是指向固件 BIOS 的 0xFFFFFFF0(32 位 CPU)位置，也就是说，BIOS 中的指令代码比 TPM 中的指令代码更早地获得系统的控制权。但是按照 CRTM 的要求，CRTM 应该包含 CPU 上电执行的第一条指令，要将 CRTM 设计到 TPM 芯片中，必须改变现有计算机体系架构和 CPU 设计，显然这在目前还不现实。

既然 CPU 上电后执行的第一条指令包含在 BIOS 中，则 BIOS 最早执行的那部分代码最适合被设计成 CRTM，如图 6.3 所示。由于 CRTM 是信任链的起点，因此 CRTM 必须是无条件可信任的，为了保障这一点，必须采取软件措施和硬件物理措施对 BIOS 中的 CRTM 进行多重保护，防止 CRTM 被篡改或破坏。

此外，BIOS 作为系统固件，还需要负责对其后加载执行的软件代码(如 MBR/IPL 代码)进行完整性度量，将信任传递下去。

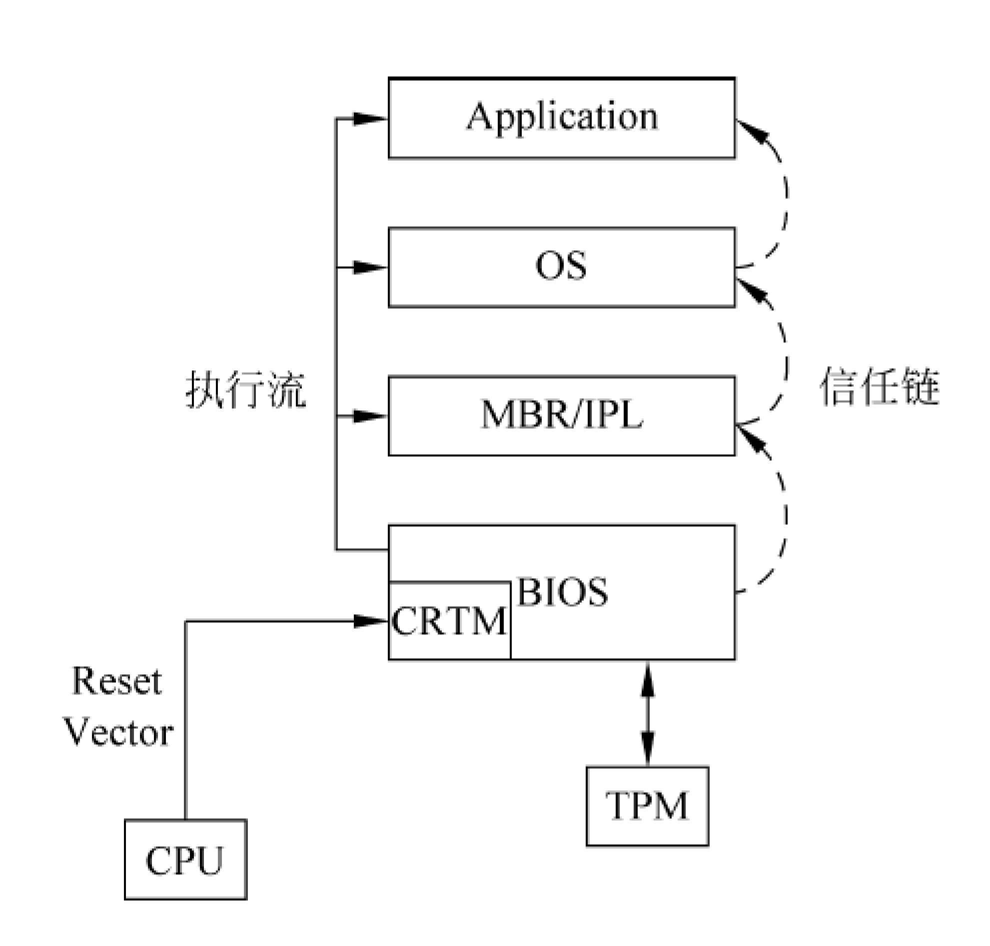


图 6.3　CRTM 包含在 BIOS 中的信任链

6.3　固件安全需求分析

本节从固件面临的安全威胁和 TCG 可信计算需求两个方面分析固件的安全需求，定义可信固件概念和安全功能。

本书前面将 BIOS 安全威胁按照来源不同分为两类：内部威胁和外来威胁。BIOS 功能障碍、配置漏洞、信息泄露属于 BIOS 内部威胁，而 BIOS 物理攻击、BIOS 恶意代码则属于外来威胁。

BIOS 功能障碍和信息泄露属于正常软件或程序模块存在的缺陷，这类威胁只需打补丁即可解决。BIOS 的配置属于 BIOS 系统的数据的一部分，配置不合理、配置错误，或配置信息丢失可能导致计算机系统某些硬件设备不能使用，或者导致系统不能正常启动。

对 BIOS 的物理攻击会破坏 BIOS 代码的完整性，使得计算机系统不能正常引导和启动，达到拒绝服务攻击目的。

嵌入到 BIOS flash 中的恶意代码会在 BIOS 或操作系统运行过程中被激活，从而对计算机系统的安全造成危害，使得系统丧失服务能力，或者导致在进入操作系统

后，危及操作系统的安全可靠运行和用户的敏感信息。

以上针对 BIOS 的安全威胁，具有一个共同的特征，即都是通过破坏 BIOS 代码或数据的完整性实现对 BIOS 系统和计算机系统的攻击或破坏。

因此，固件安全需求的第一条，是固件系统代码和数据的完整性保护要求。

TCG 可信计算要求从可信硬件开始，从底层到上层软件，每一步骤的执行都通过信任传递，建立可信链。这个过程强调的同样是实体的完整性保护。信任传递是通过完整性度量来实现的。完整性度量能够阻止内部、外部的非可信代码的加载执行，保证 BIOS 只执行来自可信任的 BIOS 厂商、设备驱动厂商或可信任用户的代码。BIOS 是完整性度量的起点，完整性度量的核心根 CRTM 是 BIOS 最早执行的代码。BIOS 需要对三种代码进行完整性度量：一是 BIOS 中除 CRTM Code 外的其他 BIOS Code；二是被 BIOS 从外部加载的 Option Rom 和其他 BIOS 层的驱动和应用程序；三是 OS Loader Code。

因此，固件安全需求的第二条，是作为可信计算的核心测量根，完成对操作系统执行前环境的可信度量过程。

当由于不可预知的故障或攻击导致 BIOS 部分代码或数据完整性遭到破坏时，BIOS 系统自身必须具备安全可靠的失败自恢复机制，以有效应对拒绝服务类型的攻击。实施这种失败自恢复机制的 BIOS 代码必须受到硬件保护，保证失败自恢复机制自身不会遭到破坏。恢复过程中同样需要对所提供的恢复内容进行可信度量，这个过程称为可信恢复。

因此，固件安全需求的第三条，是固件的可信恢复能力的要求。

定义 6-4　可信固件　可信固件是指能够提供自身完整性保护能力，包含可信计算的核心度量根，负责完成操作系统执行前环境的可信度量过程，并具有对自身进行可信恢复操作能力的固件系统。

这里有必要讨论完整性的内涵，以明确可信度量和完整性保护的目标。

信息的完整性关注信息的真实性（Authenticity）、正确性（Correctness）和可靠性（Reliability）。信息的完整性能够检测出对源信息未经授权的修改，或保障在信息的传输过程中，从源出发地发出的所有信息（包括头部数据、内容等），到达目的地后是

未被篡改的。这里的信息既包括系统的配置数据、待加工数据，也包括可执行的代码。信息的完整性同时也包括对信息起源的证明。信息的完整性是信息系统能够提供可靠服务的必要条件。

因此，可信度量既包括对信息内容的正确性测量，即信息是未被篡改过的，也包括对信息起源的证明，即信息真正的发布者与信息所指明的发布者是相符合的，信息发布者是可靠的、可以信赖的。

6.4 经典安全模型分析

安全策略(Security Policy)是一组用来控制系统如何管理、保护及分配敏感信息的规则和操作[92]。从工程的角度来看，安全策略是指对安全需求的规范描述，这种描述可以是形式化的，也可以是非形式化的。安全模型(Security Model)是系统要实施的安全策略的正式表达，安全模型必须定义一组控制系统如何管理、保护和分发敏感信息的规则集合[92]。安全模型的目的是准确地表达安全需求。

从 20 世纪 70 年代开始，伴随信息安全的发展研究，先后提出多种安全模型，一些安全模型还得到了形式化证明。这些安全模型通常是各种安全操作系统开发的安全基础和依据。相比较操作系统来说，尽管固件的功能和执行环境都要简单得多，但是现存的一些安全模型并不一定能够满足固件的安全开发需求。本文首先对典型的安全模型进行研究，然后针对固件的安全威胁和安全需求分析结果，提出适合可信固件开发的安全策略模型。

6.4.1 安全模型的分类比较

从 1973 年 D. Elliott Bell 和 Leonard J. LaPadula 提出第一个可以用数学方法证明的安全系统模型[83,84,85]以来，在随后三十多年的发展中，信息安全界先后提出了十多种具有典型性和重要影响的安全模型。此模型主要形成于 20 世纪 70 年代到90 年

代。这些模型的提出对安全操作系统的开发和各种应用系统、网络系统的安全开发提供了良好基础和指导性。根据不同模型关注的安全目标和安全控制方式的不同，本文对这些模型进行分类的比较分析，如表 6.1 所示。

表 6.1　安全模型的分类

模型分类	模　　型	作　　者	年　代	安全目标
访问控制模型	BLP 模型	D. Eliott Bell and Leonard J. LaPadula	1973	机密性
	HRU 模型	M. H. Harrison, W. L. Ruzzo, and J. D. Ullman	1976	机密性
	Biba 模型	K. J. Biba	1977	完整性
	Clark-Wilson 模型	David D. Clark and David R. Wilson	1987	完整性
	中国墙安全模型	D. Brewer and M. Nash	1989	机密性
	TE 模型	W. E. Boebert and R. Y. Kain	1985	政策中立
	DTE 模型	R. O. Brien and C. Rogers	1991	政策中立
	RBAC 模型	R. S. Sandhu	1996	政策中立
信息流模型	格模型	Dorothy D. Denning	1976	机密性 完整性
	无干扰模型	J. A. Goguen and J. Meseguer	1982	机密性 完整性

访问控制模型是现实公文世界保密管理机制的数学抽象，这种模型试图通过控制系统的行为来保证系统的安全性，访问控制也是比较成熟的系统安全技术。信息流模型[93,94,95]的目的是控制系统内部的信息流，阻止未经许可的信息流。因此，信息流模型研究的是信息在系统各实体中的流动以及由此所产生的系统实体间的相互关系。与访问控制模型相比较，信息流模型在本质上更抽象，实现也更困难，因而很少

有实用的系统采用信息流模型。

BLP 模型[83,84,85]是第一个、也是最经典的机密性访问控制模型，是根据军事安全策略设计开发的，其安全级别的划分建立在政府和军事信息分类分级的基础上，被认为是目前最成功的机密性策略模型。BLP 模型及其变体模型几乎在每个安全操作系统中都得到实现。同时，BLP 模型也是 TCSEC 标准的理论基础。

Biba 模型[96]是第一个以完整性保护为目标的安全模型，它基于完整性级别的层次格来实现完整性保护，其数学描述与 BLP 模型类似，但是，Biba 模型最大的问题是它不是源于实际的安全需求，而是建立在对安全模型进行数学分析的基础上，缺乏应用背景，这使得 Biba 模型难以在实际中得到应用。但是 Biba 模型的完整级约束的思想在系统的配置策略中却被广泛使用。

Clark-Wilson 模型[97]是目前最能反映完整性保护需求的模型，它来源于久经考验的商业方法，针对的威胁是由于操作失误或者商业欺诈而导致的商务交易受损，因此 Clark-Wilson 模型比 Biba 模型更实用。

中国墙安全模型[98]是以机密性保护为目标的商用模型，模型的基本思想是想防止企业雇员访问利益冲突双方的信息。这个模型特别适用于专业化的金融领域，例如，证券分析师可以为很多公司工作，但是，每次当分析师获取某公司内部敏感信息时，该模型阻止他同时为同行业的其他公司工作，因为同行业的公司之间存在利益冲突。该模型在证券分析师和同行业的其他公司之间装上一堵“中国墙”。

TE[99]和 DTE[100]模型通过对主体和客体的聚类，压缩了传统的访问控制矩阵的规模，降低了系统存储和搜索访问权限的代价，提高了访问决策效率。其访问权限可以灵活配置，以实现多种不同的安全目标，因此得到了广泛的应用。严格来说，TE 和 DTE 更像是一种安全实现机制，而非安全模型。

RBAC 模型[101,102]通过角色划分实现用户、会话、角色、权限的动态配置和访问控制，可用于表达不同的安全策略。从实际应用看，直接用角色来限制主体对客体的访问对大多数应用来说，只是一种粗粒度的控制。

HRU 模型[103]给出的是一个基于访问矩阵的模型系统，但该模型具有很弱的安全性质，很多关心实际问题的安全策略往往会落入 HRU 模型中的不可确定的情

况中。

尽管这些安全模型以往是被应用于安全操作系统的开发中，但是深入探讨它们实现安全保护的理论依据和安全机制，分析它们的不足，针对可信 BIOS 运行环境和安全需求在一个或多个模型的基础上进行改进或综合，对形成适合于指导可信 BIOS 开发的安全策略模型，还是具有借鉴和指导意义的。

6.4.2 BLP 模型

D. Elliott Bell 和 Leonard J. LaPadula 在 1973 年提出的 BLP 模型是第一个形式化的安全系统数学模型，1976 年，Bell 和 LaPadula 在研究报告[85]中给出了 BLP 模型改进的完整表述。BLP 模型能有效地防止信息从一个高安全级流向低安全级，主要应用在军事安全领域等具有严格安全等级划分的领域中。

BLP 模型是一个状态机模型，它形式化地定义了系统、系统状态以及系统状态间的转换规则；系统状态和状态转换规则的限制和约束依赖于模型所制定的一组安全特性。对于一个系统而言，如果它的初始状态是安全的，并且所经过的一系列规则转换都遵循该组特性，那么可以证明系统是安全的。

BLP 模型所使用的主要模型元素包括：

主体：Si，Si∈S，S 是主体的集合；

客体：Oj，Oj∈O，O 是客体的集合；

访问属性：x，x∈A，A 是访问属性集；A＝{e，r，a，w}，e 表示执行；r 表示只读；a 表示只写；w 表示读写；

访问：(Si，Oj，x)∈b，b 是所有当前访问的集合；

客体层次结构：H，H 是一个或多个树形层次结构的集合，表示客体之间离散的或层次的关系；

自主访问矩阵：M，矩阵元素 Mij 表示主体 Si 对客体 Oj 的访问属性；

安全敏感级：表示主体或客体的密级，如非密、机密、秘密、绝密，通常用整数表示；

安全类属：表示主体允许访问的客体类型范围；

安全等级：(安全敏感级，安全类属集合)的序偶对；安全等级之间建立一种称为“支配”的偏序关系，用≧表示。安全等级(L，C)≧(L′，C′)当且仅当 L≧L′ 并且 C′⊂C；支配关系具有自反性、反对称性和传递性的数学性质；

等级函数：f={(fS，fO，fC)}，fS (Si)表示主体 Si 的最大安全等级，fO(Oj)表示客体安全等级，fC(Si)表示主体当前访问安全等级；

系统状态：v=(b，M，f，H)。

在此基础上，BLP 模型定义了三种安全特性。

1. 简单安全特性(ss-特性)

当且仅当符合下面的条件时，状态 v=(b，M，f，H)和所有访问(S，O，x)∈b 是满足简单安全特性的：

x=e 或 a，或者

x=r 或 w，并且 fS(S) ≧fO(O)

2. 星号安全特性(*-特性)

当且仅当符合下面的条件时，状态 v=(b，M，f，H)和所有访问(S，O，x)∈b 是满足星号安全特性的：

(S，O，a)∈b => fO(O) ≧ fC(S)

(S，O，w)∈b => fO(O) = fC(S)

(S，O，r)∈b => fC(S) ≧ fO(O)

由于支配关系具有传递性，因此由 *-特性可以推导出下面的结果：

(S，Oj，a)∈b 并且(S，Ok，r)∈b => fO(Oj)≧fO(Ok)

3. 自主安全特性(ds-特性)

当且仅当符合下面的条件时，状态 v=(b，M，f，H)和所有访问(S，O，x)∈b 是满足自主安全特性的：

$(Si, Oj, x) \in b => x \in Mij$

其中，ss-属性和 * -属性属于强制访问控制(Mandatory Access Control，MAC)，ds-属性属于自主访问控制(Discretionary Access Control，DAC)。ds-属性是为了解决现实中对应的 need to know 的问题。

BLP 基本安全定理 如果系统的初始状态是安全的，并且系统状态的每一次迁移都满足 ss-属性、* -属性和 ds-属性，则系统的每一个后继状态都是安全的。

BLP 模型的安全目标是防止敏感信息的泄漏，其安全属性表达的安全思想简单地可以概括为“读低写高”，即主体只能读取安全级别低于(包括等于)自身被赋予的安全级别的客体，只能改写安全级别高于(包括等于)自身被赋予的安全级别的客体。由于这种安全级别的划分和安全思想直接来源于军事和政府保密的客观现实世界的要求，因此在对保密性保护要求较高的安全信息系统中得到了广泛的应用。可以说，BLP 模型是严格保密性安全模型的典范，有着不可取代的历史作用和长盛不衰的生命力。

6.4.3 BiBa 模型

借助于 BLP 模型安全级的概念，1977 年 K. J. Biba 提出第一个解决计算机系统完整性安全属性问题的模型，称为 Biba 模型[96]。

Biba 模型的完整性是在系统子集上评估的，系统子集是指那些根据函数或优先级分离出来的系统主体和客体所组成的子集。Biba 模型将完整性威胁划分为内部威胁和外部威胁两种：如果子系统中的一个组件是恶意的或不正确的，那么就会产生内部威胁；如果一个子系统试图通过提供错误信息或错误的调用函数来篡改另外一个子系统的状态，那么就会引起外部威胁。Biba 认为内部威胁可以通过程序的测试和校验解决，Biba 模型只处理外部威胁。

Biba 模型描述的基本元素包括：

S：主体集合；

O：客体集合；

I：完整性级别集合；

il：定义主体和客体完整级别的函数；

leq：定义 I 上偏序“低于或等于”的关系；

min：返回 I 的某子集中最大下界的函数；

o：主体查看(observe)客体的授权关系；

m：主体修改(modify)客体的授权关系；

i：主体调用(invoke)客体的授权关系。

Biba 模型支持 5 种不同的完整性安全策略：

1．低水标策略(The Low-Water Mark Policy)

在该策略中，每个主体的完整级别是动态、单调、非递增变化地。在任何时刻，主体的函数值 il(s)都反映了该主体前一状态的最低点。主体的低水标是主体能够查看的客体的最小完整级别，并且主体被限制只能修改那些完整级别小于或等于自身完整级别的客体，一个主体只能调用完整级别小于或等于自身完整级别的另一个主体。该策略形式化的表示为：

对于 S 中的任意主体 s 和 O 中的任意客体 o

$$s\ o\ o \Rightarrow il'(s)=\min(il(s),il(o))$$

对于 S 中的任意主体 s 和 O 中的任意客体 o

$$s\ m\ o \Rightarrow il(o)\ leq\ il(s)$$

对于任意两个主体 s1 和 s2

$$s1\ i\ s2 \Rightarrow il(s2)\ leq\ il(s1)$$

在该策略中，随着主体对客体的不断访问，主体完整级别保持单调非递增。

2．客体低水标策略(A Low-Water Mark for Objects)

在客体低水标策略中，主体的完整级别仍然随着对客体的读取访问而保持单调非递增动态变化，而客体的完整级别，也会随着主体对客体的修改访问而保持单调非

递增动态变化。该策略可形式化地表示为：

对于 S 中的任意主体 s 和 O 中的任意客体 o

$$s\ o\ o \Rightarrow il'(s)=\min(il(s),il(o))$$

对于 S 中的任意主体 s 和 O 中的任意客体 o

$$s\ m\ o \Rightarrow il'(o)=\min(il(s),il(o))$$

3. 低水标完整性审计策略

在该策略中，为每个主体和客体定义了一个 cl(current corruption level)完整性级别函数，cl 函数的取值规定为：

对于 S 中的任意主体 s 和 O 中的任意客体 o

$$s\ o\ o \Rightarrow cl'(s)=\min(cl(s),cl(o))$$

对于 S 中的任意主体 s 和 O 中的任意客体 o

$$s\ m\ o \Rightarrow cl'(o)=\min(cl(s),cl(o))$$

4. 环策略(The Ring Policy)

环策略为直接修改的保护策略提供了核心实现。在主体和客体的生存周期内，它们的完整级别是固定不变的，而且只允许对那些完整级别小于或等于主体的客体进行修改。由于允许读取任何完整级别的客体，系统的灵活性得到极大的提高。环策略可以形式化表述为：

对于 S 中的任意主体 s 和 O 中的任意客体 o

$$s\ m\ o \Rightarrow il(o)\ leq\ il(s)$$

对于任意两个主体 s1 和 s2

$$s1\ i\ s2 \Rightarrow il(s2)\ leq\ il(s1)$$

5. 严格完整性策略(The strict Integrity Policy)

严格完整性策略可以看成是 BLP 模型的反转模型，此策略的前两条规则直接对

应 BLP 模型的 ss-特性和 *-特性，但是增加了对调用操作的完整级别的限制。该策略的三条规则可形式化表述为：

对于 S 中的任意主体 s 和 O 中的任意客体 o

$$s\ o\ o \Rightarrow il(s)\ leq\ il(o)$$

对于 S 中的任意主体 s 和 O 中的任意客体 o

$$s\ m\ o \Rightarrow il(o)\ leq\ il(s)$$

对于任意两个主体 s1 和 s2

$$s1\ i\ s2 \Rightarrow il(s2)\ leq\ il(s1)$$

Biba 模型的严格完整性策略的基本思想可以简单地概括为“读高写低”，即主体只能读取完整性级别高于(包括等于)自己被赋予的完整性级别的客体，只能改写完整性级别低于(包括等于)自己被赋予的完整性级别的客体。

Biba 模型是经典的完整性模型，特别是它的严格完整性策略，与 BLP 模型是异曲同工的。但是它存在的最大缺点在于：完整性级别划分困难，有时即使强行划分了，也是无实际意义的，这样导致难以在具体系统中得到应用，而 BLP 模型安全级别是一种很自然的划分。

6.4.4　Clark-Wilson 模型

1987 年，David D. Clark 和 David R. Wilson 比较了计算机系统中军用安全目标和商用安全目标的不同，指出军用安全目标的重点是控制敏感信息的泄漏和分级访问，而商用安全目标则更关注信息的正确性、无篡改、无错误，前者更关注信息的保密性，后者更关注信息的完整性，军用的安全策略和实现机制并不完全适用于商用系统；并根据商用实际安全需求提出了商用的完整性安全模型，称为 Clark-Wilson 模型[97]。

Clark-Wilson 模型提出以良构事务(well-formed transaction)和职责分离(separation of duty)为核心的安全策略。良构事务要求用户程序不能够随意处理系统数据，而必须受限于某些操作规则以保障数据的内在一致性。职责分离要求每个

操作被分成多个步骤，并且每个步骤由不同的用户来完成，以保障数据的外在一致性。特别地，通常要求由一个用户创建良构事务，而由另一个不同的用户来执行该事务，创建者不能执行自己创建的良构事务。

系统只包含两种完整性级别的数据：受限数据项（Constrained Data Items，CDIs）和非受限数据项（Unconstrained Data Items，UDIs）。CDIs 是在系统状态转换过程中完整性必须得到保障的数据项，UDIs 的完整性则不受系统保护。系统允许 UDIs 在合法规则的操作下转换成 CDIs。系统日志属于只允许追加写操作（append-only）的 CDIs。

系统中定义了两类过程：完整性验证过程（Integrity Verification Procedures，IVPs）和转换过程（Transformation Procedures，TPs）。IVPs 用于验证在某系统状态下 CDIs 的完整性。TPs 都是良构事务，负责将 CDIs 从一个有效状态转换到另一个有效状态。系统中只允许通过 TPs 对 CDIs 进行操作和转换。

Clark-Wilson 模型定义了 9 条完整性保障规则以保证该模型安全策略的有效实施，包括 5 条认证规则（Certification rules，C1-C5）和 4 条执行规则（Enforcement rules，E1-E4）。认证规则由系统管理员、安全管理员等实施，以保证 TPs 和 IVPs 是符合完整性策略的；执行规则由计算机系统强制实施。

C1：IVP 必须确保所有的 CDIs 处于有效状态中。

C2：TP 必须确保对任一个 CDI 的操作结果仍然是一个有效的 CDI，安全管理者必须指定 TP 和 CDIs 的初始操作关系列表（TPi，（CDIa，CDIb，CDIc，…））。

E1：只有经过认证的 TP 才能操作 CDI，系统必须维持（TPi，（CDIa，CDIb，CDIc，…））关系列表。

以上三条规则保障 CDIs 的内在一致性。

E2：系统必须维持一个访问三元组列表（UserID，TPi，（CDIa，CDIb，CDIc，…）），用户只能通过被授权执行的 TP 访问 CDI。

C3：E2 中的访问操作必须满足职责分离要求。

E3：系统必须对请求执行 TP 的用户进行身份认证。

以上三条规则保障 CDIs 的外在一致性和职责分离策略。

C4：所有 TPs 的行为必须被记录到日志 CDIs 中以允许对其操作进行重构。

C5：TP 对 UDI 操作的结果，要么是将其转换为一个 CDI，要么将其遗弃。

E4：只有系统特定的认证管理员才能修改关系操作列表。特别地，认证管理员不具有执行管理列表中任何 TPs 的权限。

图 6.4 总结了 Clark-Wilson 模型中这些规则是如何对系统中的数据进行完整性保护的。

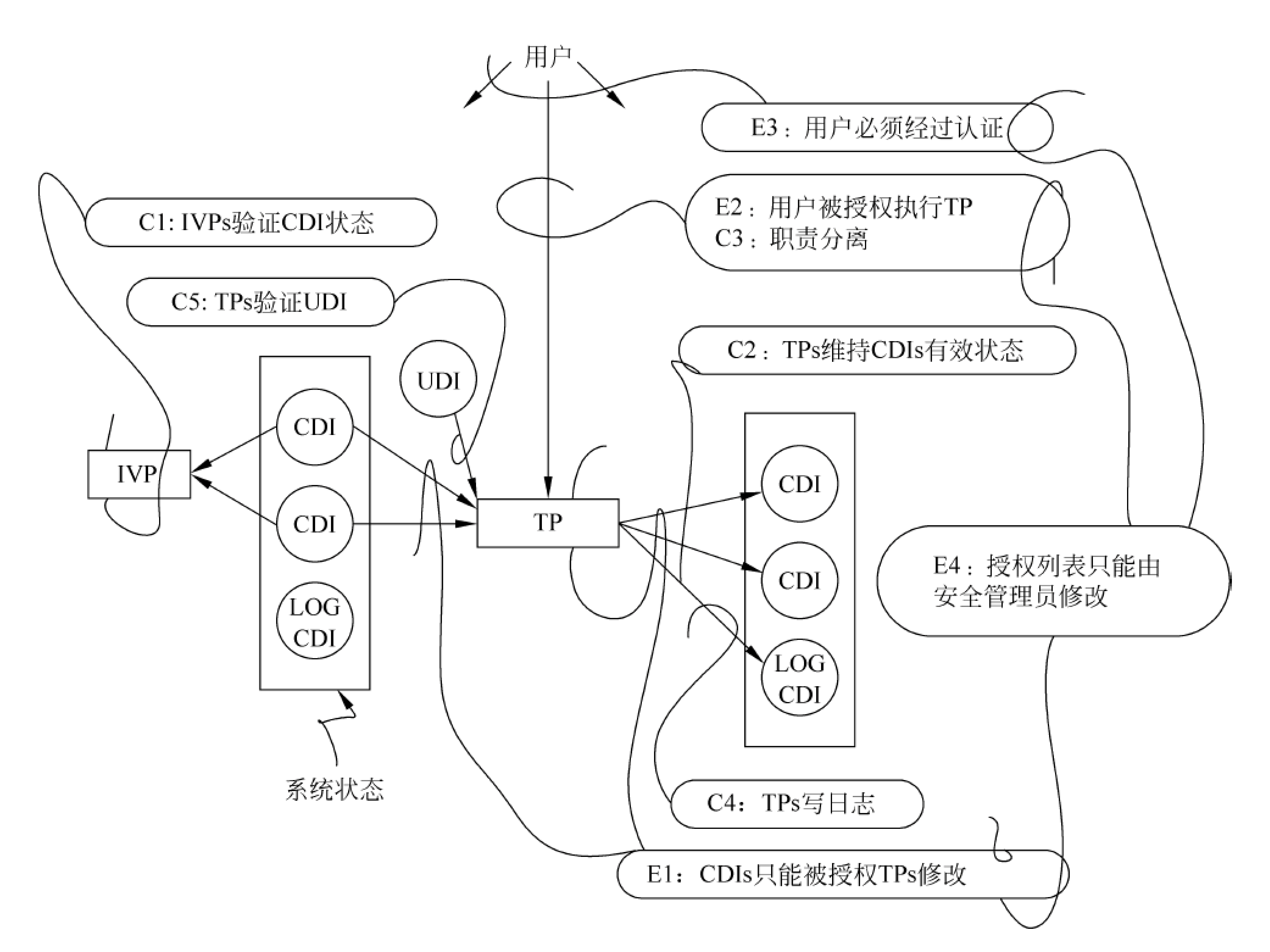


图 6.4　Clark-Wilson 模型的完整性规则

Clark-Wilson 模型被认为是实现了全部三个完整性保护目标的安全模型[104]：IVP 和 TP 验证了数据的内在和外在的一致性，访问权限列表阻止了非授权用户对数据的修改，职责分离防止了授权用户对数据的非法修改。

同 Biba 模型相比较，Clark-Wilson 模型来源于久经考验的商业方法，这种方法已

在现实世界中使用了很多年且被证明是行之有效的，因此 Clark-Wilson 模型比 Biba 模型更实用。

6.5 可信固件的保护环模型

固件系统不同于操作系统，它不像操作系统一样是一个多用户、多进程的系统，而且固件操作系统同用户的交互，也一般只是通过 BIOS setup 模块来实现对系统和平台参数的一些配置。但是，由于固件系统处在底层，操作系统需要依靠固件系统来装载引导，如果固件系统受损，则操作系统也无法引导运行，计算机系统将变得不可使用。

对固件系统的保护，通过本书前面的分析可知，主要是固件系统的完整性保护。一方面，保护固件系统的代码是完整的，阻止入侵的恶意代码的执行，防止对系统代码篡改造成的拒绝服务攻击；另一方面，保护固件系统配置数据的完整性，防止一些关键的配置数据不正确而造成固件系统无法启动运行，或者导致某些关键设备不可使用。

以 Clark-Wilson 模型为基础，本节综合 Biba 模型严格完整性策略的"读高写低"思想、Clark-Wilson 模型的"良构事务"和完整性验证思想，针对固件系统的代码和数据类型特点，对固件系统的代码和数据划分了完整性级别，提出了对固件的代码和数据进行完整性保护的可信固件保护环模型(Protected-Ring Model)。该模型同 Clark-Wilson 模型相比较可实现性更强。

在该模型中，划分了两类数据项：受保护数据项(Protected Data Item，PDI)和不受保护数据项(Non-protected Data Item，NDI)。即在固件系统中，数据集合 DATA 为：

PDS＝Set of PDI
NDS＝Set of NDI
DATA＝PDS ∪ NDS

PDS 是固件正常运行所需要的最小数据的集合，如果 PDS 中存在任何一项错误的数据，则会导致固件不能正常运行和引导操作系统。PDS 中主要包含跟平台芯片、

外围固定设备控制器的设置参数以及固件查找引导记录的顺序等重要数据。NDS中的数据则不是那么重要，即使存在错误也不会影响固件正常运行和操作系统引导，这些数据如系统时间、日期以及其他一些同固定硬件平台无关的配置信息。

保护环模型中，将固件环境中的代码划分为3类：核心可信代码(Core Trusted Code，CTC)、普通可信代码(Ordinary Trusted Code，OTC)和非可信代码(Un-Trusted Code，UTC)。即固件系统代码CODE为：

CTS=Set of CTC
OTS=Set of OTC
UTS=Set of UTC
CODE=CTS ∪ OTS ∪ UTS

核心可信代码是固件系统的基础代码构成部分，是固件系统能够正常运行和加载引导操作系统的那些代码。普通可信代码是运行于固件平台上的附加的应用程序、驱动、协议以及其他第三方厂商或用户定制工具。CTC和OTC要求能够通过代码自身包含的信息实现对代码生产者或拥有者的认证。非可信代码是不能被证明拥有合法生产者或用户身份的用户代码、第三方工具代码或其他恶意代码等。

CTS、PDS以及CRTM构成一个最小可执行的固件系统。

定义允许的操作集合：

OPERATION={r,m,e}

操作r允许对数据进行读取，操作m允许对数据进行修改(包括追加)，操作e允许代码被执行(包括从一个代码中调用另外一个代码)。

定义代码完整性认证函数、代码可执行认证函数和代码类属认证函数如下：

AI(c)=0, c ∈CODE　　并且c是不完整的；
AI(c)=1, c ∈CODE　　并且c是完整的；
AE(c)=e, c ∈CODE　　并且c是允许执行的；
AE(c)=¬e, c ∈CODE　　并且c是禁止被执行的；
AC(c)=CT, c ∈CTS；

$AC(c)=OT, c \in OTS;$

$AC(c)=UT, c \in UTS;$

可信固件的保护环模型核心策略是通过保护固件系统运行所需要的关键代码和数据的完整性，保有一个最小可用固件系统。模型的保护策略是通过以下 8 条强制性实施规则(Mandatory Enforcement Rules，MER1-MER8)来保障的。

MER1：只有完整的可信代码才允许被加载执行，即

$$\forall c \in CODE\ AE(c)=e \Rightarrow AI(c)=1 \text{ and } (AC(c)=CT \text{ or } AC(c)=OT)$$

MER1 规则阻止非可信代码的执行，如嵌入到固件系统中的恶意代码；MER1 规则同时也阻止完整性被破坏的可信代码的执行，以避免内容被篡改的可信代码的执行对系统造成损害。

MER2：只有核心可信代码才被允许修改受保护数据项，即

$$\forall c \in CODE\ \forall d \in PDS\ c\ m\ d \Rightarrow AC(c)=CT \text{ and } AI(c)=1$$

如果允许普通可信代码对受保护数据项进行直接修改，可能会使修改后的数据项内容不合适而导致固件系统不能正常运行。例如，一个经过认证的普通用户的固件应用程序可能会试图改变系统引导顺序，或者修改硬盘参数，而普通用户可能由于对系统平台不够了解而使这些参数引起冲突或不能适合于硬件平台，从而导致固件系统不能正常运行。

MER3：CRTM 维持并保护 CTS 和 PDS 列表集合。

恶意用户可能通过固件系统外部的技术手段对 CTC 代码类属进行篡改，导致系统对一个本来是 $c \in CTS$ 的核心可信代码 c 进行认证时，$AC(c) \neq CT$，从而造成对系统的拒绝服务攻击。CRTM 可以通过自身维护的 CTS 列表来获知这一篡改，并在 CTC 受到此种篡改时做出正确的反应，如启动可信恢复。

MER4：用户必须经过认证才能使用 CTC 或 OTC 访问或修改系统数据。

某些 CTC 允许用户通过交互方式对系统的 PDI 进行修改，如 BIOS setup 程序。因此，对这些交互用户进行身份认证是系统安全重要的一个环节。

MER5：在保证最小系统可用性的前提下，允许安全管理员(Security Administrator，SA)对 CTC、OTC、UTC 进行转换，或对 PDI、NDI 进行相互转换，并对 CRTM 中的 CTS 和 PDS 列表集合作相应的维护。

这条规则在保障系统安全的前提下，赋予系统更大的灵活性，以适应硬件平台的临时变动和安全管理、应用的操作需求的变化。

MER6：系统中所有对代码的可信测量、对用户的认证和对 PDI 的修改操作必须记录到日志中。

MER7：CRTM 负责提供对 PDI 修改访问的仲裁保护，并在 CTC 和 PDI 完整性受到破坏时，启动可信恢复过程。

一旦加载执行的 OTC 可能会违反规则试图对 PDS 中的 PDI 进行修改操作，CRTM 能够制止这种修改；当发现某个或某些 CTC 内容或身份被篡改而造成系统不能继续安全运行时，CRTM 要及时对系统作可信恢复。

MER8：CRTM 必须受到物理的硬件保护。

通过以上 8 条规则，保护环模型能够很好地对固件系统的完整性和可用性实施保护。

图 6.5 是保护环模型的环结构示意图。

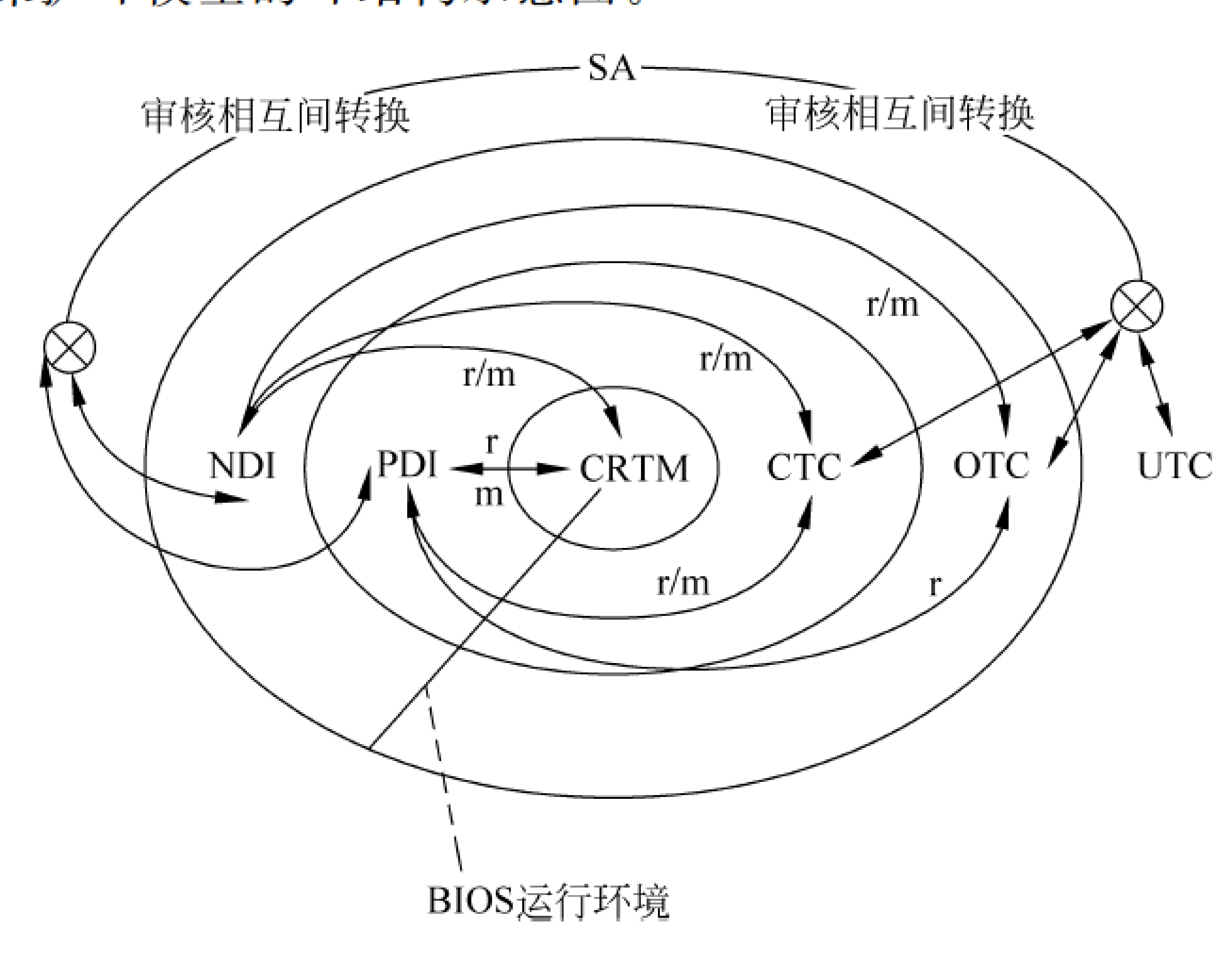


图 6.5　可信固件保护环模型的环结构

最内层环是 CRTM，受物理硬件保护，只能通过外部物理手段进行更新。CTS 和 PDS 构成第二层环，其中包含的 CTC 和 PDI 完整性被破坏后会触发 CRTM 的可信恢复机制；第一层和第二层构成固件系统的最小可执行系统；OTS 和 NDS 构成第三层环，环中 NDS 包含的 NDI 不受保护，即使被破坏也不影响固件系统的正常运行和引导操作系统，环中 OTS 包含的 OTC 完整性被破坏时，会影响部分非关键或附加功能的正常应用，但不妨碍 BIOS 系统正常运行和引导操作系统；环外的 UTC 由于不可信，所以即使被嵌入到 BIOS flash 中也不会被执行，如嵌入固件中的木马、恶意代码或其他未经认证的第三方工具程序和增值代码等。这三层环构成固件系统的动态运行环境。而 UTC 属于不可信代码，不会被加载到固件执行环境中执行。

当由于硬件平台的变化、服务和管理需求变更导致需要对 PDS 和 NDS 中的数据项进行调整，或者需要将新开发的增值服务程序嵌入到固件系统中时，需要特定的安全管理员审核调整需求，按照 MER5 规则调整相关项并更新固件系统。

假设保护环模型的初始状态是可信的。在系统执行过程中，MER1～MER8 能够保护代码和数据的完整性不会受到破坏，因此系统运行的随后状态都是可信的。当外部手段导致除 CRTM 外的其他部分（CTC、PDI）完整性被破坏时，或者有非可信代码进入时，如操作系统下的用户可能通过软件修改固件内容或者向固件中嵌入新的代码等，CRTM 能够有效地恢复系统并保证恢复后的系统是可信的，因此系统的可用性得到保障。

6.6　本章小结

从 2003 年至今，TCG 的可信计算成为安全界的热点，出现了较多以可信完全代替安全的论调。本章通过考较在不同历史阶段可信和安全两个概念的使用情况和含义，得出可信和安全的本质是行为安全的结论。指出 TCG 在实现规范上与其可信概念定义之间存在的差距：规范实现的是一种静态信任关系的度量，而非可信概念定义的行为度量。因此可以说按照 TCG 规范实现的可信计算只是在现有技术条件下

能够实现的有限程度的安全,是安全的一种权宜之计,而非安全的终极目标。

本章内容是安全固件开发的理论基础。传统固件 BIOS 的设计和实现没有考虑安全问题。第 5 章讨论了现有传统固件 BIOS 的安全保障问题,本章试图设计一种新的安全固件产品。为此分析了安全固件的安全需求:固件自身保护需求;固件之后的 OS Loader 验证需求;失败自恢复需求,提出适合于安全固件开发的可信固件保护环策略模型。



第 7 章

可信度量基础与度量方法

TCG(Trusted Computing Group)是成立于2003年的一个非盈利性的国际工业标准组织,其前身是1999年由IBM、Intel、Microsoft等IT巨头发起成立的TCPA(Trusted Computing Platform Alliance)组织。TCG组织的目标是开发、定义和促进基于硬件的可信计算和安全技术的开放性统一标准。TCG的可信规范涵盖九个方面的内容,即基础结构、PC终端、服务器、软件栈、可信网络连接、移动设备、存储设备、可信平台模块,硬拷贝设备。TCG可信计算以可信平台模块(Trusted Platform Module,TPM)为信任根,定义了一个由可信实体构造的分层结构的可信计算生态环境。

本章对TCG可信计算相关规范进行介绍和分析,在这些规范的基础上,研究建立同TCG规范相符合的可信固件可信度量的方法和机制。

7.1 可信计算平台

可信计算平台是可信计算生态环境中的计算单元,典型的可信计算平台是可信计算机。

7.1.1　可信计算机参考结构

可信计算机的突出特征，是在硬件上增加了可信计算模块 TPM 作为平台信任的硬件基础。TCG 组织推荐的可信计算机参考逻辑结构[32]如图 7.1 所示。

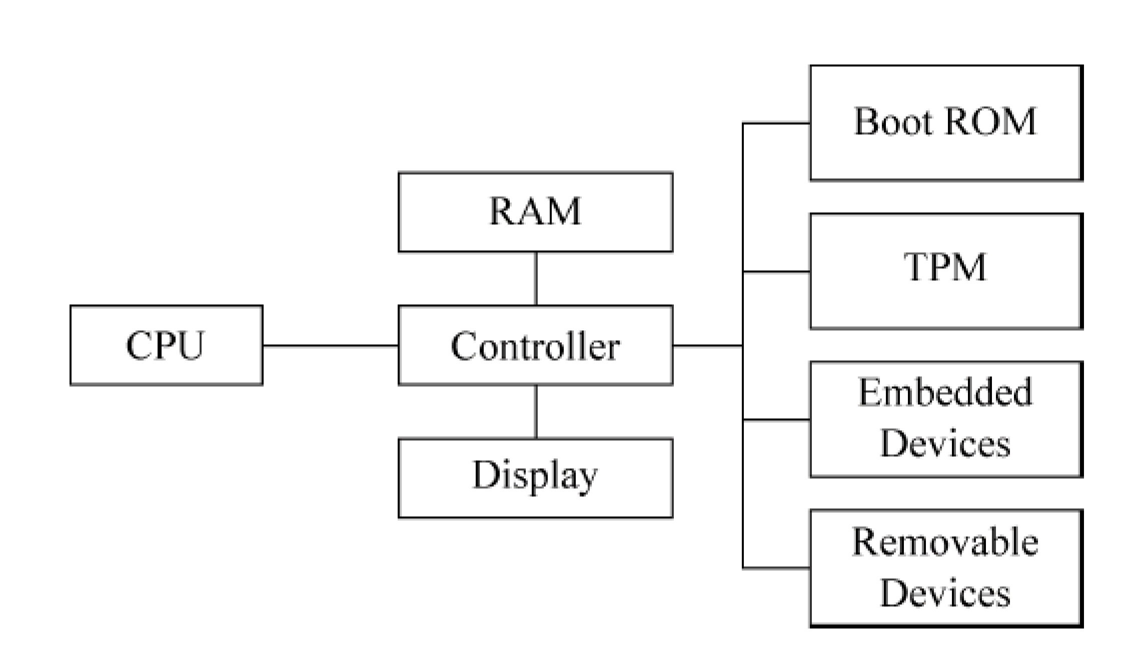


图 7.1　可信计算机参考结构

TCG 的可信计算机参考结构无需改变现行计算机的硬件组织架构，只需要通过外围总线将 TPM 连接到主板上即可。目前通用的做法是将 TPM 设备通过 LPC 总线连接到主板上的南桥芯片上，见本书第 2 章图 2.1、图 2.2 示例。由于 TPM 芯片是信息安全的核心部件，不可避免受到各国安全政策和标准的管制，因此主板厂商通常只在主板上保留一个 TPM 插座，而不直接将 TPM 芯片焊接到主板上。

7.1.2　可信平台的基本特性

可信平台至少应该包含三个基本特性：保护能力、验证能力、完整性度量和报告。

保护能力提供对敏感信息和敏感信息存储部件的访问保护，使得这些信息只能通过独立的一组安全命令来存取。如对存储在 TPM 中的完整性测量值的保护、密钥的存储保护等。

验证能力是指平台能够证实信息或实体的真实性的能力，如 TPM 的身份证明、

平台身份证明。

完整性度量过程是指收集平台数据和配置信息并将信息摘要保存到 TPM 中的 PCR 中；完整性报告则是对 PCR 中的完整性测量值进行验证的过程。

以上三个基本特性保障可信平台能够抵抗篡改攻击，证明平台代码和数据的完整性和真实性，从而安全地执行代码并保护敏感数据的机密性。通过这三种基本特性，各平台间能够建立起信任关系，实现可信计算生态环境的构建。

7.1.3 信任根和信任链

信任根是可信平台中必须被无条件信任的部件。可信平台包含三种信任根：可信度量根(Root of Trust for Measurement，RTM)、可信存储根(Root of Trust for Storage，RTS)和可信报告根(Root of Trust for Reporting，RTR)。

RTM 是完整性度量的计算引擎，同时也是信任链传递的起点。RTM 作为可信的代码是主板平台启动后最先执行的代码。启动代码随着平台的启动，通过一种“信任传递”的过程将信任扩展到整个平台。在“信任传递”过程中，前一阶段的代码在把控制权传递给下一阶段前，先测量其度量值，并把度量值安全地保存起来，这个过程反复持续下去，直到可信的操作系统启动。

RTS 是维护完整性度量值的正确性和度量顺序的计算引擎。RTS 保护所有委托给 TPM 的密钥和数据。RTS 基本上是指所有密钥管理功能，包括密钥产生、密钥管理、加密和解密。

RTR 是能够安全可靠地对 RTS 存储的信息进行报告的计算引擎。RTR 允许经过验证的挑战者获取受 TPM 保护的区域中的数据，包括平台配置寄存器(PCRs)和非易失内存。并用签名密钥签名证实这些数据的真实性。其中平台配置寄存器不仅保存数据，而且记录数据被保存的次序。

从 RTM 开始到系统固件、到操作系统引导程序、到操作系统、再到应用程序，计算机执行通过一级认证一级、一级信任一级的方式，把这种信任扩展到整个计算机系统，实现从硬件到软件，从底层到上层的信任传递，构成了计算机系统运行的信任链，

其中信任链的信任根由物理安全和管理安全确保。

7.1.4 可信平台模块

可信平台模块(Trusted Platform Module,TPM)是可信计算的核心设备,是可信平台的硬件基础。TPM是一个含有密码运算部件和存储部件的小型片上系统。作为可信平台的可信硬件基础,TPM能够提供可信计算平台的三种基本特性,是可信平台三种可信根的硬件基础。TPM的结构[105]如图7.2所示。

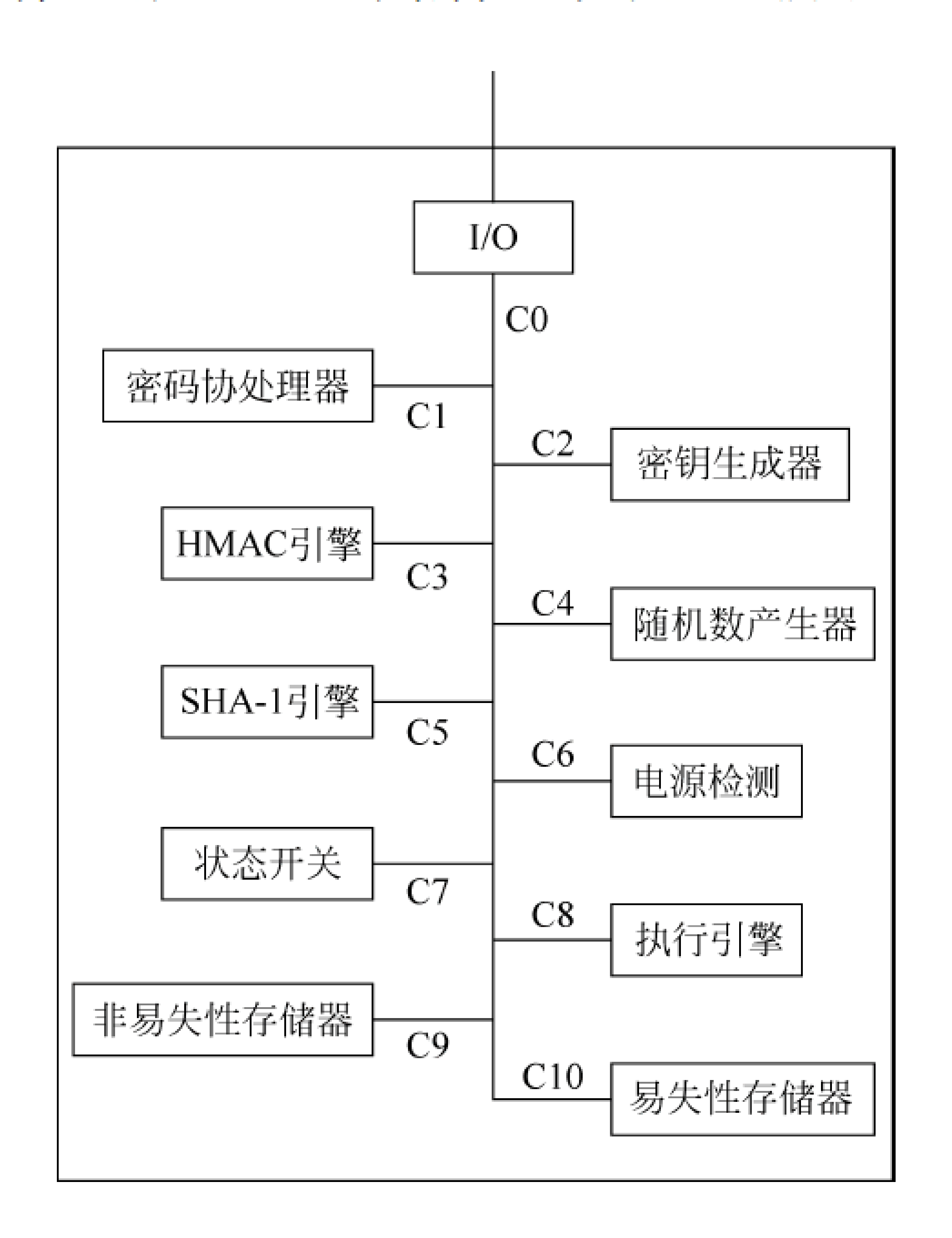


图7.2 TPM的部件结构

输入/输出组件C0:管理着通信总线上的信息流。它为外部和内部总线的通信执行协议编码/解码,并向合适的组件发送消息。I/O组件执行Opt-In开关组件和需要访问控制的TPM函数的访问策略。

密码协处理器C1:实现TPM中的密码操作,这些操作包括非对称密钥生成

(RSA)、非对称加密/解密(RSA)、散列(SHA-1)、随机数生成(RNG)。TPM 使用这些能力生成随机数和非对称密钥,对存储数据签名和加密。TPM 也可以实现其他的非对称算法,使用不同的算法进行签名和封装。密码协处理器也包含对称加密引擎,TPM 使用对称加密对认证信息进行加密,在传输会话中提供机密性。规范建议 RSA 密钥长度最少为 2048 位。

密钥生成器 C2:创建 RSA 密钥对和对称密钥。

HMAC 引擎 C3:执行 HMAC 计算。HMAC 的计算遵照 RFC2104 中的定义,密钥的长度(RFC2104 中的 K)必须为 20 字节。分组大小(RFC 2104 中的 B)必须为 64 字节。

随机数产生器 C4:它是 TPM 中随机数的来源。TPM 使用这些随机数值来生成密钥和签名所需的随机数。TPM 在每次调用时能够提供 32 位的随机数。

SHA-1 引擎 C5:散列能力主要被 TPM 自身使用,是散列算法的可靠实现。TPM 向外部提供散列接口以支持在平台导入阶段进行度量,以便在有限能力的计算环境下能够使用 TPM 的散列函数。TPM 实现 FIPS-180-1 中定义的 SHA-1 散列算法,SHA-1 的输出为 160 位。

电源检测 C6:管理 TPM 的电源状态。TCG 要求 TPM 能够接受计算机所有电源状态的变化通知,而且电源检测也支持物理在线声明。在平台操作受到物理限制期间,TPM 可以限制命令的执行。在 PC 中,操作的限制出现在上电自检(POST)期间,这时往往要求操作员通过键盘输入声明物理在线。在受限制执行模式或导入状态,TPM 可以允许访问某些命令。

状态开关 C7:提供 TPM 上电/断电、使能/非能和激活/非激活的机制和保护状态。开关组件维护永久和易失标志的状态,并执行与这些标志相关联的语义。这些标志的设置需要 TPM 所有者的授权或平台上物理在线的声明。远程实体在没有 TPM 所有者知识或操作员对该平台没有物理在线的情况下,不能改变 TPM 的状态。

执行引擎 C8:运行程序代码以执行从 I/O 端口接收到的 TPM 命令。执行引擎是一个保证操作被适当隔离和保护区被保护的关键组件。

非易失性存储器 C9:被用来保存永久身份和与 TPM 相关联的状态。非易失

性存储器可以拥有设置项(如 EK),也可以由被 TPM 所有者授权的实体分配和使用。

易失性存储器 C10:即动态 RAM 存储器。

在集成 TPM 设备的计算机平台上,用户或程序只能通过 TPM 提供的命令或上层封装的 API 来使用和操作 TPM。

7.1.5 可信平台典型应用场景

以可信平台为基础,计算机和网络环境中的计算机可以互相验证对方的软硬件状态,从而为解决信息安全目前的困境带来新的希望。可信平台典型的应用包括如下几方面。

1. 风险管理

风险管理的目标是最大程度减少由于恶意破坏或意外造成的公司和个人资产的丢失和泄密。TCG 的保护存储技术能够减少信息资产损失的风险。保护存储可以保护非对称加密的公私钥和对称加密的密钥,这种保护以硬件为基础、以密封存储和计算为保障,避免由于密钥泄漏而造成信息资产的非法访问和泄漏。

2. 资产管理

资产管理的目标是防止未经授权对计算资产进行访问和使用,以及防止资产丢失被窃。其中资产追踪是资产管理的一项重要手段。TPM 可用于对平台拥有者身份和平台硬件的唯一标示,并确保这种标示不会物理地被清除或替换。如果资产失窃,偷窃者也不能访问资产中的信息,因此信息不会被泄漏,而偷窃者也不能从信息资产中获利。

3. 电子商务

电子商务需要客户和销售商互相信任,并尽可能避免非法第三方对交易的干扰。

可信平台的完整性报告和身份证明能力为电子商务提供了坚实的信任基础。

4. 安全监控和应急响应

IT管理人员通常要花大量的时间对付病毒和黑客攻击，应急响应队伍要能够对存在安全漏洞的系统进行快速隔离和漏洞修复和补丁。在这个过程中，管理人员需要花费较长时间扫描企业或公司网络中各个计算机的系统配置，以决定哪些系统需要更新或修补。

TPM可用于确保每台计算机都能够真实地报告自身的配置信息和参数。在平台引导启动过程中，TPM度量各软硬件部件，并将度量结果存储在TPM的PCRs中。应急响应人员可以利用这些度量值来确定哪些计算机存在问题，易于受到病毒或黑客的攻击。安全管理人员也可以基于这些度量值决定拒绝那些处于不安全状态的计算机连接到网络中。

7.2 可信平台中的证书分析

可信计算技术的基础是密码技术，尤其是公开密码技术，因此在可信平台中使用了多种数字证书。本节对可信平台中使用的证书作简单的介绍和总结。

TCG规范定义了5种类型的证书[32,108]：背书证书(Endorsement Key Certificate)、平台证书(Platform Certificate)、身份证书(Attestation Identity Key Certificate，AIK证书)、符合性证书(Conformance Certificate)和验证性证书(Validation Certificate)。

7.2.1 TPM背书证书

TCG规定EK的私钥部分是不向外界泄漏和不可以被TPM外部访问的，并且从不用来加/解密用户数据。为了达到标准要求，需要对产生、植入EK过程以及EK的载体TPM的来源严格把关。

EK 证书通常由 TPM 的生产厂商签发。基于隐私考虑，目前 EK 证书仅在向 Privacy-CA 申请 AIK 证书时提供身份证明，之后就由 AIK 来替代 EK 提供 TPM 身份证明。EK 证书中包含了 EK 公钥，EK 公钥虽然是公开的，但由于是鉴别 TPM 身份的唯一证据，所以也是秘密和敏感的。

为了保护平台、平台拥有者、平台用户的隐私，EK 证书在平台上存放时应该受到保护，只有平台拥有者或经过其授权的实体才能访问 EK 证书。

表 7.1 列出了 EK 证书必须包含的域。

表 7.1　EK 证书必须包含的域

域　名	内容描述
版本号	证书遵循标准的版本号，通常为 3(即 X.509 version 3)
序列号	签发者分配给证书的唯一的标识编号
签名算法	签发者对证书签名采用的签名算法
签发者	EK 证书的签发者名称
有效期	证书的有效期
主题	通常为空，签发者需要使用扩展的主题替代名来描述
公钥信息	描述 EK 公钥使用的算法和公钥值
扩展信息	关于证书的附加属性
证书策略	描述证书签发的多个策略项
主题替代名	描述使用目录名字形式的 TPM 制造商、TPM 模型和版本号的相对甄别名(RDN)
基本限制	描述签发者是否为 CA
主题目录属性	包含诸如 TPM 支持的算法、符合的规范版本号等信息

7.2.2　平台证书

平台证书主要用于确认某个具体的特定平台的制造者和描述平台的属性，用来声明、证实一个集成有 TPM 的计算平台符合 TCG 规范。通常平台证书由平台集成制造商签发。由于该证书里包含了 EK 证书的索引，出于隐私的考虑，平台证书的使

用也受到严格的限制，只限在向 Privacy-CA 申请 AIK 证书时使用。其中 EK 证书索引起到关联平台与 TPM 的作用。

为了保护平台、平台拥有者、平台用户的隐私，平台证书在平台上存放时应该受到保护，只有平台拥有者或经过其授权的实体才能访问平台证书。

表 7.2 列出了平台证书必须包含的域。

表 7.2　平台证书必须包含的域

域　　名	内容描述
版本号	证书遵循标准的版本号，通常为 2(即 X.509 version 2)
序列号	签发者分配给证书的唯一的标识编号
签名算法	签发者对证书签名采用的签名算法
持有者	指向平台的 TPM EK 证书的引用
签发者	平台证书的签发者名称
有效期	证书的有效期
属性	描述平台符合的 TCG 规范、平台的 CC 认证保护轮廓标识符、CC 认证安全目标标识符等信息
扩展属性	关于证书的附加属性
证书策略	描述证书签发的多个策略项
主题替代名	描述使用目录名字形式的平台制造商、平台模型和版本号的相对甄别名(RDN)

7.2.3　TPM AIK 证书

AIK 证书被用来鉴定对 PCR 值进行签名的 AIK 私钥的真实性，它包括 AIK 的公钥和其他发布者认为有用的信息。

AIK 证书是由一个可信的、能够校验各种证书和保护客户端隐私的 Privacy-CA 来签发的。在完整性报告和平台验证的过程中，通常使用 AIK 私钥代替 EK 私钥进行签名，既证明了签名 TPM 的合法性，又避免了暴露用户和平台的隐私信息。AIK 证书中包含的 AIK 公钥则可以用来验证 TPM 签名的真实性。

表 7.3 列出了 AIK 证书必须包含的域。

表 7.3　AIK 证书必须包含的域

域　　名	内容描述
版本号	证书遵循标准的版本号，通常为 3(即 X.509 version 3)
序列号	签发者分配给证书的唯一的标识编号
签名算法	签发者对证书签名采用的签名算法
签发者	AIK 证书的签发者名称
有效期	证书的有效期
主题	证书名称，通常为空
公钥信息	描述 AIK 公钥使用的算法和公钥值
扩展信息	关于证书的附加属性
证书策略	描述证书签发的多个策略项
主题替代名	包含来自 TPM 的 EK 证书主题替代名中的 TPM 制造商、模型、版本号内容；包含平台制造商、模型、版本号内容；TPM 身份标签
基本限制	描述签发者是否为 CA
主题目录属性	包含 TPM 符合的规范、平台符合的规范、TPM 支持的算法、TPM 的 CC 保护轮廓标识、TPM 的 CC 安全目标标识等信息

符合性证书是用来声明、证实某一类计算平台的实现符合 TCG 的哪些规范、符合哪些安全要求。它不能用来唯一标识一个平台，而是针对某一类的平台。该证书可由平台或 TPM 制造商签发，也可由任何具备规范符合性测试能力的第三方测试机构签发。

验证证书用来表明平台上某一个组件的指令或功能具有证书中所包含的属性，证书中通常需要包含用于验证的测试数据。这些组件可以是平台上的任何软硬件部件，如视频卡、存储卡、内存控制器、网络控制器、CPU、键盘、鼠标以及各种系统和应用软件。验证证书可以由组件制造商或第三方验证机构签发。

在可信平台中，所有证书均采用 ISO/IEC/ITU-T X.509 格式。

讨论：Privacy-CA 可以是一个公共的 CA，也可以是一个企业内部专有的 CA，这需要根据可信平台使用的范围来确定。如果只在企业内部使用，则可以是一个专有的 CA，如果在广泛的不定范围内运行，则需要一个公共 CA。但是，当在一个广泛的不定范围内使用时，如何规定可信任的 CA，如何相信 CA 提供的证书都是真实可信

的，这个问题的解决具有很大的困难和不确定性。

另外，在平台信任链的构造过程中，代码度量是否采用类似的身份证书，TCG 没有涉及这个问题，也没有规定信任链构造的度量方法，这些问题需要产品开发者自己决定，显然这不利于可信平台的统一推广使用。

7.3 TPM 密钥分析

作为计算和通信终端的硬件可信基础，TPM 保存和管理着多种密钥，以实施平台的身份认证、签名和数据加密功能。

7.3.1 TPM 密钥类型

TCG 定义了 7 种密钥类型[32]。

1. 签名密钥(Singing Key)

签名密钥是非对称密钥，主要用于对应用数据和消息进行签名。签名密钥既可以是可移植的，也可以是不可移植的。可移植的密钥允许在不同的 TPM 设备之间导入/导出。

2. 存储密钥(Storage Key)

存储密钥是非对称密钥，主要用于加密存储数据或其他密钥。

3. 身份证明密钥(Identity Key)

TPM 的身份证明密钥通常是指 AIK(Attestation Identity Key)密钥，是一种不可移植的签名密钥。当需要提供 TPM 的可信证明时，就使用 AIK 私钥对来对数据

值进行签名，如在完整性报告的过程中使用 AIK 私钥对 TPM 提供的 PCRs 的值进行签名。

4. 背书密钥(Endorsement Key)

也称 EK 密钥，是不可移植的非对称密钥。对一个 TPM 以及一个平台而言，EK 是唯一的。有两种方式产生 EK，一种方式是在生产过程中在 TPM 内部产生 EK；另一种方式是将外部产生的 EK 在 TPM 生产过程中导入。

EK 与 TPM 是一一对应的，如果 EK 用来签名，那么任何实体都可以跟踪 EK 的使用。出于安全和隐私保护方面的考虑，并不直接使用 EK 来进行数据的加密和签名。EK 的主要功能是生成身份证明密钥 AIK 和建立 TPM Owner。

5. 包扎密钥(Binding Key)

对称密钥，通常用于对平台之间的交换数据进行加密传输。

6. 遗赠密钥(Legacy Key)

在 TPM 外部创建的可移植密钥，导入到 TPM 中后可用于签名和加解密操作。

7. 认证密钥(Authentication Key)

对称密钥，用于对 TPM 参与的传输任务的数据加密保护。

7.3.2　TPM 密钥管理

TPM 芯片内部只有少量有限可用的易失性和非易失性存储空间，然而，可信平台的应用可能需要保存大量的密钥和受保护数据。因此，TCG 定义了 TPM 外部缓存空间和缓存管理器。

密钥块(Key Blob)用于在 TPM 以外的存储空间存储 TPM 相关的密钥。密钥块是经过 TPM 加密的，并且对 TPM 来讲是非透明的(即 TPM 知道该密钥块的存在

并保留关联信息)。密钥块可以存储到 TPM 以外的任何存储设备上,如 flash 盘、硬盘、网络文件服务器等。通过密钥块内容的摘要值、密钥块句柄或者是密钥块的全局唯一性标识符可以实现对密钥块的引用操作。

TPM 提供命令接口以支持外部程序管理 TPM 内部的有限存储空间,这种外部程序称作密钥缓存管理器(Key Cache Manager,KCM)。KCM 负责 TPM 内部和外部存储设备上的密钥交换和移动操作,为此 KCM 需要跟踪 TPM 内部密钥存储空间的活动情况以及 TPM 密钥使用情况,以及完成对外部存储设备上的密钥块进行索引、读写操作。通过 KCM,非激活的密钥在加密保护后允许被移出 TPM 芯片以便腾出 TPM 存储空间交换进其他需要激活的密钥。存储在外部设备中的密钥由于被 TPM 加密保护,因此在 TPM 外部无法用来进行签名和加解密操作,只有移入 TPM 内部成为激活密钥才可以用于签名和加解密。

只有两个密钥是固定嵌入在 TPM 中并且不可以被移出 TPM 芯片的:背书密钥(Endorsement Key,EK)和存储根密钥(Storage Root Key,SRK)。SRK 在建立 TPM 的拥有者时产生,该密钥的私钥保存在 TPM 的保护区域中,需要提供授权数据才可以使用。SRK 的私钥不能被导出,后续生成的密钥都直接或间接的由该密钥加密,它是 TPM 密钥存储体系的根。当清除 TPM 的拥有者时,SRK 也同时被清除,所有被 SRK 直接或间接加密的数据都将无法使用,包括密钥和数据。

图 7.3 显示了可信平台基于 TPM 的密钥管理和存储结构。

7.3.3 AIK 及其证书生成安全分析

EK 是 TPM 中最核心的密钥,出于安全和隐私保护的考虑,不直接使用 EK 来进行数据的加密和签名。EK 的主要功能是生成身份证明密钥和建立 TPM Owner,由 TPM Owner 来生成存储根密钥 SRK,使用 SRK 来加密存储其他密钥。

AIK 可以看做是 EK 的别名。建立 TPM Owner 后就可以创建 AIK,TPM 中 AIK 的数量理论上是不受限制的,因此可以减少因为 AIK 统计信息而导致的隐私泄漏。

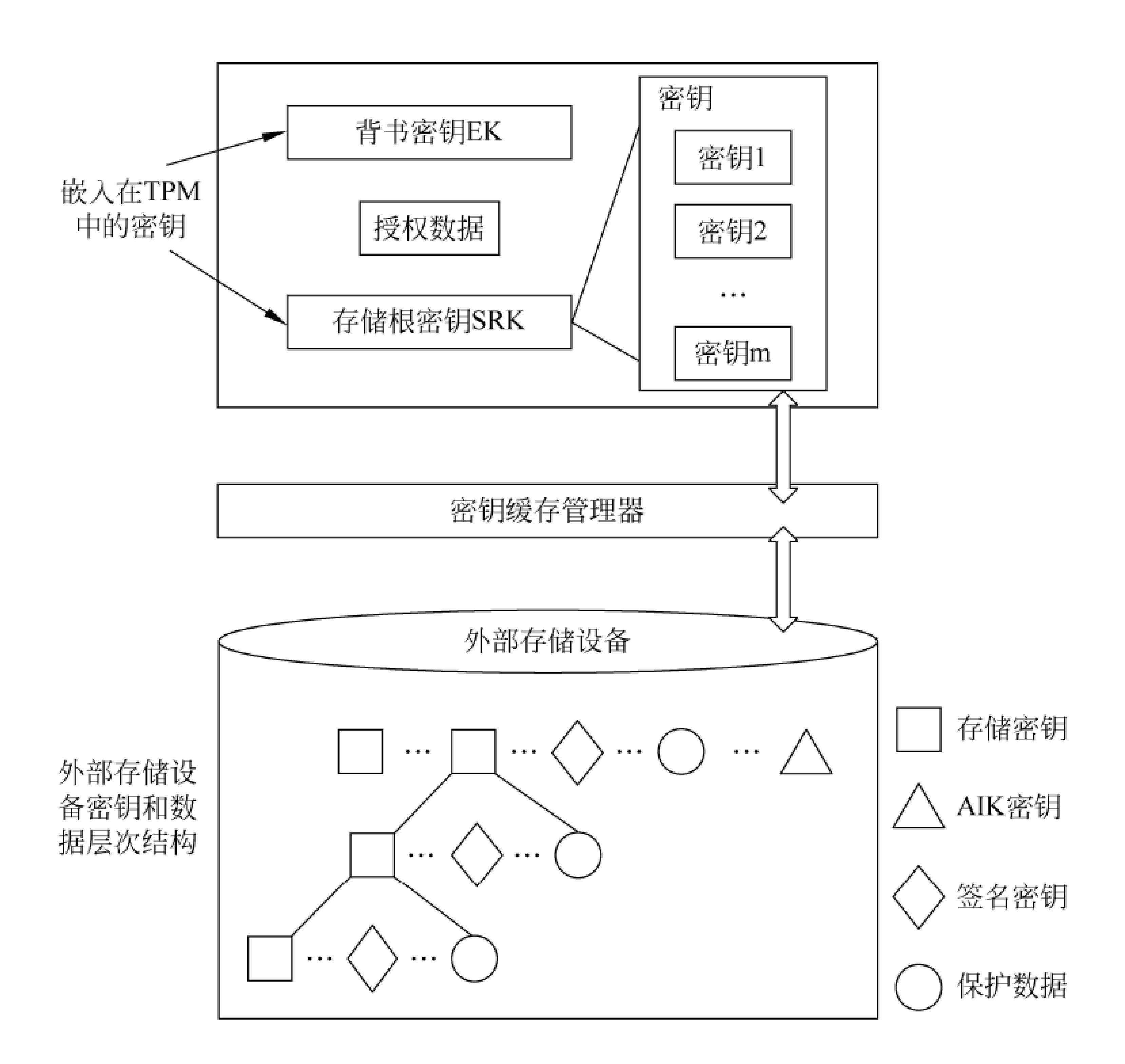


图 7.3　TPM 密钥管理和存储结构

AIK 的生成虽然使用了 EK，但是生成的 AIK 中却不包含任何有关平台或 EK 的隐私信息。这就使得 AIK 可以证明 TPM 的身份但不会泄漏任何隐私信息。AIK 只用来代替 EK 进行签名，不能用于数据加密，并且 AIK 只能用于对 TPM 内部产生的数据进行签名，如 PCR 值、TPM 的其他密钥、TPM 状态信息等。AIK 不能用于对 TPM 外部产生的数据进行签名，以免攻击者借此伪造 PCR 的值。

AIK 既可以代替 EK 进行平台证明，又不会泄漏 TPM 身份的隐私信息，因此，AIK 和 AIK 证书的产生过程和安全性就十分重要。下面分析 AIK 和 AIK 证书产生的过程及其安全性。

AIK 和 AIK 证书产生过程可以分为 3 个阶段。

第一阶段　AIK 生成和证书请求：

① TPM Owner 使用 TPM_MakeIdentity 命令生成一对 AIK 公私钥对，长度至少是 2048 位，将 AIK 的私钥使用 SRK 加密保存。

② TPM 构造 TPM_IDENTITY_CONTENTS 结构，结构中包含刚生成的 AIK 公钥以及 TPM 自身的标识信息，如 EK 证书和平台证书。

③ TPM 使用刚生成的 AIK 私钥对 TPM_IDENTITY_CONTENTS 结构进行签名，将签名值和结构内容一起发送给 Privacy-CA，等待 Privacy-CA 接受请求后生成 AIK 证书。

第二阶段　Privacy-CA 签发证书：

④ Privacy-CA 接受请求，检验签名信息是否正确，若正确则根据 TPM_IDENTITY_CONTENTS 中的 AIK 公钥生成 AIK 证书。

⑤ Privacy-CA 产生一个 session key(对称密钥)，使用这个 session key 对刚生成的 AIK 证书进行加密。

⑥ Privacy-CA 使用发送申请请求的 TPM 的 PUBEK(EK 公钥)对 session key 加密，构造一个 TPM_SYM_CA_ATTESTATION 结构，结构中包含加密了的 session key、加密了的 AIK 证书以及一些加密算法参数等。

⑦ Privacy-CA 发送 TPM_SYM_CA_ATTESTATION 结构到 TPM。

第三阶段　TPM 激活证书：

⑧ TPM 接收到 PM_SYM_CA_ATTESTATION 结构，使用 TPM_ActiveIdentity 命令对结构进行解密。该命令首先使用 TPM 的 PRIVEK(EK 私钥)解密加密证书的 session key，然后再使用 session key 解密 AIK 证书。

至此，AIK 和 AIK 证书的产生过程结束，这个 AIK 即可以使用。

对该过程的安全性分析：

(1) 使用 AIK 私钥部分对 TPM_IDENTITY_CONTENTS 结构进行签名是为了防止结构的内容在传送过程中被篡改。

(2) Privacy-CA 在产生 AIK 证书后，使用 TPM 的 PUBEK 对 AIK 证书进行加密后再传送给 TPM，这样只有拥有与该 PUBEK 对应的 PRIVEK，才能解开这个加密了的证书，从而使用该 AIK。这样既证明了发送请求的 TPM 的真实性，防止了第

三者冒充 TPM 发送请求,也使得其他窃听者即使得到了这个加密的证书也无法完成解密,无法使用该 AIK。

(3) Privacy-CA 在这个过程中的安全性至关重要,Privacy-CA 必须保证不会泄露任何敏感信息,也不会在 AIK 证书产生过程中欺骗 TPM。

(4) Privacy-CA 不立即检查收到的请求的合法性,可能会由于大量冒充的无效请求造成对 Privacy-CA 的拒绝服务攻击。

(5) 攻击者可能通过同时替换 TPM_IDENTITY_CONTENTS 结构中的 AIK 公钥和 TPM_IDENTITY_CONTENTS 结构的签名实现该结构内容的篡改并成功欺骗 Privacy-CA,使得 Privacy-CA 签发无效的 AIK 证书。为了避免这种情况的发生,本书认为,应该在过程③中使用 PRIVEK 对 TPM_IDENTITY_CONTENTS 结构进行签名。

7.4 TCG 完整性度量与报告要求

TCG 可信平台的信任链和可信计算环境的信任边界是通过完整性度量和报告机制建立并逐步扩展的,TCG 在其规范中给出了指导性建议[32,109]。

7.4.1 完整性度量要求

完整性度量内核负责完整性度量过程。一次完整性度量过程称为一次完整性度量事件。一次完整性事件包含两类关键数据:度量数据和度量值。度量数据是需要被度量完整性的数据信息或程序代码;度量值是由度量内核根据度量数据计算产生的消息摘要值。这两类数据独立存放:度量数据存储在度量内核能够访问到的任何存储区域,而度量值则需要在 TPM 内部由 TPM 保护存储。

一次度量事件要产生度量日志(Stored Measurement Log,SML)。度量日志和度量值要求能够反映度量事件的发生顺序。所有的度量值通过 Hash 叠加的

方法计算得到一个公用的摘要值并存储到 TPM 中的 PCR 寄存器中(Platform Configuration Register)。PCR 中的值在系统重新复位后会被清空。PCR 值 Hash 叠加的方法为:

PCR[n]=SHA-1(PCR[n] || 度量值)

其中||表示连接操作。即把一次完整性事件计算产生的度量 Hash 值,同第 n 个 PCR 寄存器中的当前值进行连接操作,然后对连接操作的结果计算 SHA-1 消息摘要,得到第 n 个 PCR 寄存器的新值。TPM 中定义有多个 PCR 寄存器,分别记载不同类型的完整性度量事件的度量值。

SML 的数据量会很大,因此不会被存储到 TPM 中。如果 SML 的内容被篡改,可以通过 PCR 中的值检测出这种篡改。因此,PCR 的度量值能够用于对 SML 的完整性进行验证。

7.4.2 完整性报告要求

可信报告根 RTR 有两种基本功能:一是向外部实体报告可信平台完整性度量的结果,二是基于可信平台的身份验证报告结果的真实性。

请求报告的平台称为挑战方,需要向挑战方报告自身完整性的平台为报告方。完整性报告的过程为:

① 挑战方向报告方发出报告请求,请求报告方的 PCRs 配置结果;

② 报告方的报告代理程序(Agent)从报告方平台收集 SML;

③ 报告代理向报告方平台的 TPM 请求 PCRs 的值;

④ TPM 使用 AIK 私钥对 PCRs 的值进行签名,发送给报告代理;

⑤ 报告代理获取报告方平台证书,将平台证书、签名的 PCRs 值、SML 返回给挑战方;

⑥ 挑战方通过 PCRs 值的签名验证报告方的真实性,根据 SML 记录重新计算度量 Hash 值,通过将计算结果与 PCRs 的结果比较,判断报告方的完整性。

根据完整性报告的结果，挑战方决定是否信任报告方。

TCG给出了完整性度量与报告的指导性原则，但是并没有规定可信报告的方法。由于平台涉及硬件部件和多层软件系统，统一可信度量方法对于可信平台的构造和使用推广是很必要的。

7.5　可信固件的可信度量方法

可信固件的保护环模型，通过代码的完整性和真实性来保证数据的完整性和正确性。因此，可信固件的保护环模型的可信度量，主要是针对代码的度量。

代码度量的一种方法，是对代码本身不做任何的封装要求，即代码同传统BIOS环境下的代码是相一致的，代码本身不包含任何可信度量的相关信息。在用户初次使用平台时，BIOS不能够进行可信度量，而需要用户确认对BOIS和相关硬件进行可信初始化，之后才能在BIOS的启动和运行过程中进行可信度量。这种可信初始化，就是在初次使用时，产生BIOS环境下各代码消息摘要，BIOS对这些消息摘要进行存储保护，以便在以后的过程中通过消息摘要的比较判断各代码的完整性，从而实施可信度量。

这种方法存在两个明显的缺点：一是可信初始化后，BIOS可加载的代码就固定下来，如果有新代码需要增加进来，就必须重新作整个初始化过程，因此，这种方法不利于动态地对代码进行管理，这个问题在EFI BIOS中尤其明显，因为在EFI BIOS中，允许用户临时加载驱动或应用程序；二是对于可信BIOS来讲，代码的执行需要同时满足两个条件——完整性和类属，代码的类属决定代码可以访问的数据集合。这种可信初始化方法不能满足可信度量对于代码类属的度量要求。

针对本书第6章提出的可信固件的保护环模型的强制性实施规则，本节提出由可信注册、可信封装、可信验证三个环节相结合的固件可信度量方法。该方法符合本章前面所述的TCG可信平台的信任基础和原则，同时又很好地解决了上述可信初始化过程的缺陷。

为了后续描述的方便，定义可信固件的代码单元概念。

定义　代码单元　一个代码单元(Code Cell)是指在固件环境下最小的可独立加载的代码模块。

7.5.1　可信注册

可信注册的过程，是确定代码单元身份可信和代码类属的过程。一个将要进入到固件运行环境(保护环模型的内三层)中的代码单元的持有者，需要向安全管理员SA提交其身份证明(证书或公钥)。安全管理员负责完成下面的操作：

1. 审核身份证明的有效性，确定代码类属(CT或OT)

在固件生产过程中，身份证明应该是固件生产商或平台制造商签发的非对称密钥的公钥，或者是包含公钥的数字证书。在固件使用过程中，身份证明是由SA或Privacy-CA签发的非对称密钥的公钥，或者是包含公钥的数字证书。注册的公钥用于对代码单元进行可信封装。

UTC是不能在固件运行环境中运行的，提交注册的代码单元要由SA和注册申请者(用户或固件开发程序员)共同审核需注册的代码单元的类属(CT或OT)。审核的准则遵照保护环模型的强制性实施规则2(MER2)。

2. 保存身份证明

在固件生产过程中，包括固件厂商和平台集成厂商，需要把可信注册的公钥或数字证书通过存储设备移交给平台的最终用户。在固件使用过程中，安全管理员SA首先要将被注册的公钥导入到CRTM能够访问的保护存储区域，以用于可信验证过程。关于注册公钥保护的安全必要性，在本节后面将加以阐述。

3. 维护CRTM的CTS列表

对于需要注册为CTC的代码单元，SA必须在CRTM的CTS列表集合中增加

该代码单元的标志项。注册为 OTC 的代码单元则无需更改 CRTM 的 CTS 列表集合。CTS 列表集合中对代码单元的关联引用,可以采用代码单元的消息摘要值或者全局唯一性标识符(GUID)。

只有经过可信注册的代码单元,才能够在固件运行环境中被加载运行。

7.5.2　可信封装

可信封装的过程,是代码单元持有者为其代码单元提供真实性和完整性证明信息的过程。

设有信息元素如下:

ccell : 代码单元
signature : 数字签名
digest, digest' : 消息摘要
pubkey : 注册公钥
privkey : 注册私钥

设有信息函数如下:

Hash() : 计算信息的消息摘要
Encrypt() : 使用私钥签名
Decrypt() : 使用公钥解签
Concatenate() : 两个或多个信息的连接操作
GetCCell() : 从 CTC/OTC 信息中分离出代码数据
GetSignature() : 从 CTC/OTC 信息中分离出数字签名
VerifyDigest() : 对两个消息摘要进行比较操作,0—不相等 1—相等

则可信封装的过程表示为:

1. 计算代码单元消息摘要

digest ← Hash(ccell)

2. 使用注册私钥对代码单元计算数字签名

signature ← Encrypt(digest, privkey)

3. 构造可信的代码单元

CTC/OTC ← Concatenate(ccell, signature)

构造的可信代码单元 CTC/OTC 即可以编译或嵌入到固件映像文件中，或存储在其他控制器芯片和外围存储设备中，在固件运行过程中由固件自动加载或由用户手工加载运行。

7.5.3 可信验证

可信验证的过程是对可信代码单元的证明信息进行验证的过程，即验证可信代码单元身份真实性和内容完整性。

可信验证的过程表示为：

1. 计算被度量可信代码单元 CTC/OTC 的消息摘要

ccell ← GetCCell(CTC/OTC);
digest ← Hash(ccell)

2. 使用注册公钥对被度量可信代码单元签名解签

signature ← GetSignature(CTC/OTC);
digest'← Decrypt(signature, pubkey)

3. 验证结果

if VerifyDigest(digest, digest') = 1

then CTC/OTC 是完整可信的

else CTC/OTC 是伪造的或被篡改

如果可信验证的结果为真,则被验证的 CTC/OTC 允许被加载执行。其执行时访问 PDS 或 NDS 的行为由 CRTM 根据 CTS 列表进行监督控制。

除 CRTM 外,任何需要在固件运行环境中加载运行的代码都必须经过可信注册和可信封装,因此安全管理员 SA 必须对可信度量过程可能涉及的注册信任方信任关系进行管理和维护,包括:固件厂商、平台集成商、部件制造商、OS Loader 软件商、平台用户等。

7.5.4 注册公钥的保护

考虑下面的一种攻击情形:攻击者使用自己的私钥对恶意代码单元进行签名,并使得可信固件在运行过程中能够使用对应的公钥进行解签校验,则会得到成功的可信验证结果,导致攻击者可向可信固件中植入并成功运行恶意代码。为防止此种攻击的出现,用于可信验证的解签公钥不能同被验证代码单元一同存放,而必须密封保存在 TPM 中进行保护,而且可信度量的解签操作也需要在 TPM 内部完成。当然,这种保护同非对称加密的公钥公开并不相矛盾,只是保护可信验证时使用的公钥来源是可信的,不会被恶意者替换。安全管理员 SA 负责将可信固件涉及的注册信任方的公钥导入到 TPM 中。

7.6 本章小结

本章对可信固件的开发,在按照可信固件保护环策略模型和规则设计实现可信固件安全机制的前提下,另一个重要的指导思想,就是尽量符合 TCG 可信计算的相关规范。为此本章对 TCG 可信计算机的结构、信任链、信任关系及其管理等关键内容进行了分析,在此基础上提出的由可信注册、可信封装、可信验证三个步骤构成的

可信度量方法，既满足了可信固件保护环模型的强制实施规则，又能够符合 TCG 完整性度量和报告的规范要求。

改进之处在于，这里的可信度量，不仅仅是 TCG 所规定的完整性度量，而是进一步包含了代码安全级别的度量。可信固件保护环模型通过代码的安全级别，限制了普通可信代码部分的有害行为，从而间接实现了对代码修改受保护数据项的行为度量。但尚不能解决对于更多的、更完全的安全行为度量问题。



第 8 章

可信固件的开发实现

固件产品的开发，是同平台硬件紧密结合的，尤其是同主板所使用的芯片组和外围设备控制器类型密切相关。本书基于可信固件保护环策略模型，以新一代 EFI/UEFI 固件为实现基础，构造一种可信固件产品原型 UTBIOS(Trusted BIOS based on Unified EFI)。本章论述 UTBIOS 实现的关键技术。

8.1 UTBIOS 开发软硬件平台基础

EFI(Extensible Firmware Interface)是操作系统与平台固件层的新一代模型与接口规范，由 Intel 公司在 1998 年开始开发，并首先用于 Intel 的 64 位服务器产品中。2003 年 Intel 公布了 EFI 标准和部分框架代码[5]，联合业界推广用 EFI 标准取代传统 BIOS。2005 年，业界成立统一的 EFI 标准组织(Unified EFI，UEFI)(www.uefi.org)，致力于开发、管理和推动 UEFI 固件标准和产品化进程。UEFI 最早的标准以 EFI 1.10 规范为基础，目前最新标准为 UEFI 2.1[7]。

Intel EFI 平台创新架构(The Intel Platform Innovation Framework for EFI，本

节简称 Framework)[42]是 EFI/UEF 规范的产品级框架实现,可以应用于 IA(Intel Architecture)架构的任何一种平台来构造平台固件。Intel 将 Framework 的源代码公开,并资助 CollabNet 成立网络社区(www.tianocore.org),管理和促进对源代码持续的开放性开发。

Framework 代码只是规范实现的公共框架代码,而构造计算机固件产品还需要与平台相关的代码,这些与平台相关的代码也是固件产品的核心,由硬件芯片组厂商提供代码,或仅提供资料而由固件厂商来开发平台代码,两者都需要商业授权。

由原信息产业部支持,中国电子科技集团信息化工程总体研究中心同 Intel 合作,取得 Intel 对 Intel D945gnt 主板平台代码的授权。本书的 UTBIOS 原型产品实现正是以该平台的固件框架代码和平台代码为基础。

UTBIOS 产品原型实现的硬件平台为 Intel D945gnt 和 Foxconn-945G7MA-8KS2 主板。采用 Foxconn-945G7MA-8KS2 主板是因为该款主板直接支持 TPM 芯片的拔插,且与国产 TPM 芯片结合较好。Foxconn-945G7MA-8KS2 主板同 Intel D945gnt 主板采用了相同的芯片组,只存在少量的外围设备控制器的不同,因此移植平台代码的工作比较容易实现。

UTBIOS 的开发采用国产 TPM 芯片(SINOSUN SSX35B)作为可信的硬件基础,为 UTBIOS 提供存储保护功能。兆日的该款芯片符合 TPM 1.2 规范。

8.2 UTBIOS 结构与流程设计

UTBIOS 的开发,需要对 Intel Framework 及其平台代码结构进行调整,增加相应的安全机制,使之符合可信固件的保护环模型。

8.2.1 CRTM 的安全构造

CRTM 是信任链建立的起点,包含可信度量过程最早执行的代码,是被无条件

信任的。CRTM 要满足 4 个约束条件：

（1）CRTM 自身受到物理保护，不能通过纯软件写方式更新和升级。对 CRTM 的更新、升级要么通过物理硬件操作进行，要么通过能够证明是物理现场（physical-presence）操作的软件方式进行；

（2）CRTM 具备对后续执行模块进行可信度量的能力；

（3）当可信固件其他部分完整性遭到破坏时，CRTM 具备对其进行可信恢复的能力；

（4）为了方便计算机固件的升级更新需求，CRTM 的代码应该是完成（2）和（3）中功能及其所需初始化环境的代码最小集。

在 Intel Framework 的代码中，定义了一个空的 Security 阶段（SEC Phase），以期使得 Framework 的设计与 TCG 的 CRTM 概念相吻合。但是 Intel 没有给出 SEC 阶段的实现，也没有对该阶段内容的讨论。

根据上述 CRTM 的 4 个约束条件，本节设计将 Intel Framework 的 SEC 和 PEI 阶段合并，并将 DXE 阶段中的 USB 驱动移入到 PEI 阶段，形成 CRTM Code。CRTM 主要部件包括：

- CPU、芯片组、主板、内存、堆栈的初始化代码；
- TPM Driver；
- 消息摘要代码；
- 可信恢复代码；
- PEI Core Code；
- CTS 和 PDS 集合列表。

在固件 flash 芯片中，CRTM 占据最高端的 64KB 空间（0xFFFF0000～0xFFFFFFFF），该部分空间受到软硬件双重写保护。

8.2.2　可信度量结构与流程

在 Intel Framework 中定义了框架代码执行的七个阶段[42]：

1. SEC：Security

安全阶段，在 Intel 的产品中为空。

2. PEI[110]：Pre-EFI Initialization Environment

负责完成 CPU、Chipset、主板、内存、堆栈的基础初始化工作，在这个阶段建立起 C 语言程序的执行环境。

3. DXE[111]：Driver Execution Environment

是平台设备、总线、驱动、协议执行驱动的主要阶段，平台的完整初始化和驱动在这个阶段完成。

4. BDS：Boot Device Selection

设定和选择引导设备的阶段。

5. TSL：Transient System Load

OS Loader 装入、执行，以及 OS 引导阶段。

6. RT：Runtime

操作系统运行阶段，这个阶段 OS 会调用固件的一些服务程序。

7. AL：After Life

由操作系统返回固件运行控制阶段，如操作系统崩溃、异常，需要回到固件运行阶段进行诊断。

按照可信固件安全策略和模型要求，对 Intel Framework 经过重新设计和调整后形成的 UTBIOS 结构与流程如图 8.1 所示。

在 UTBIOS 的结构流程设计中，按照 8.2.1 节的安全要求对 CRTM 进行了重

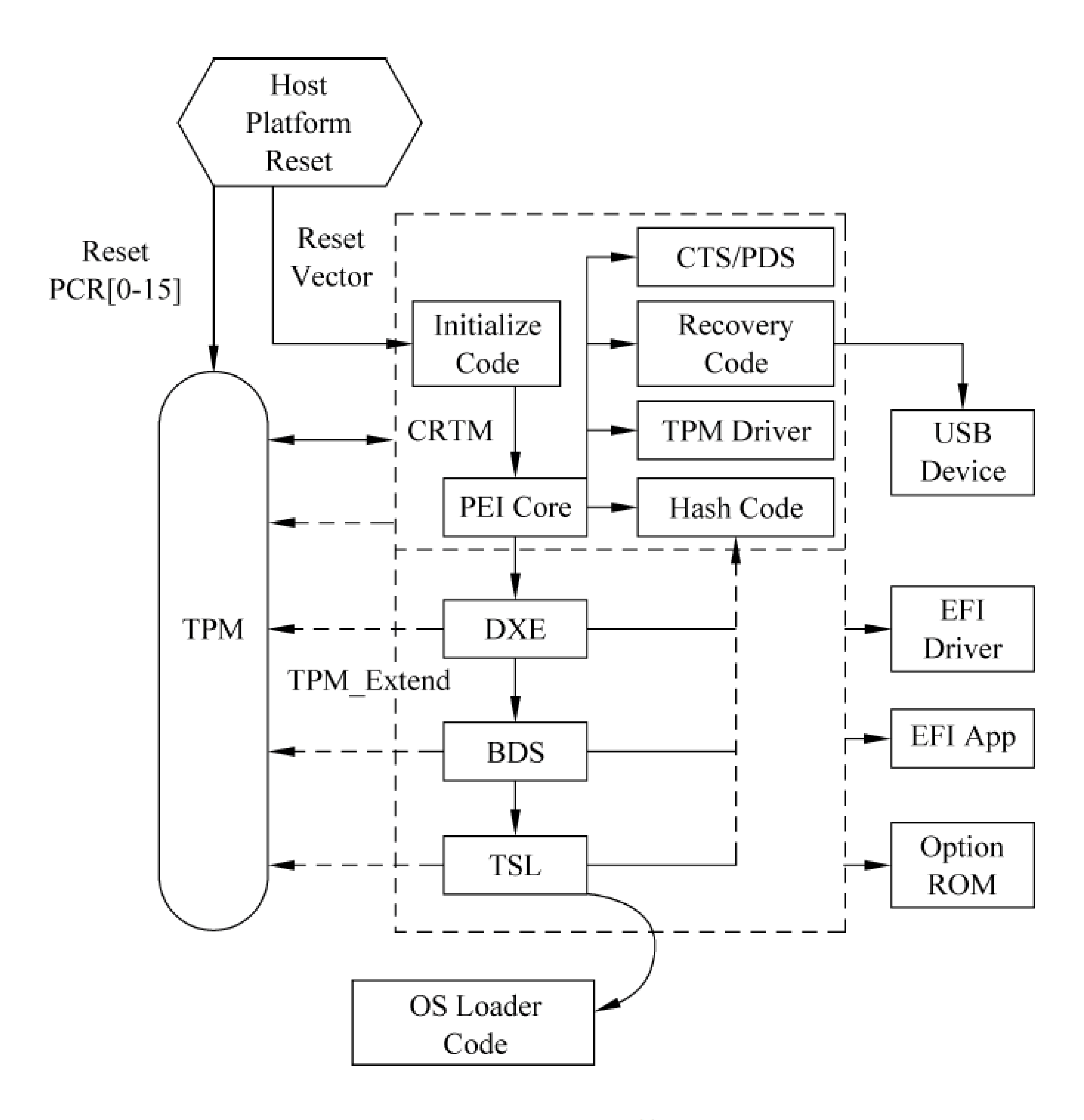


图 8.1 UTBIOS 结构流程

构。系统上电复位后，首先执行 CRTM 中的 Initialize Code，做 CPU、chipset、Memory、Stack 初始化工作，建立 C 代码执行环境；进入 PEI Core，调度 CRTM 其他代码，初始化 TPM，安装 TPM Protocol、Recovery Protocol、Hash Protocol。

随后的执行阶段，在进入下一个阶段前，或者对每个阶段中的代码单元加载准备执行前，都需要进行可信度量。可信度量的方法采用 7.5 节中描述的方法和过程。除对固件 flash 芯片中的代码进行调度外，UTBIOS 允许在 DXE、BDS、TSL 阶段根据平台和用户实际需求，从外部存储设备或芯片中调度加载 EFI 驱动、EFI 应用程序或扩展卡中的 OPROM。对这些代码的调度执行同样也需要进行可信度量。

可信度量由 CRTM 提供统一的度量实现代码，而请求度量者只需要根据度量返回的结果决定被度量代码单元的执行与否。如果遇到 CTC 度量失败，则 CRTM 会向用户请求可信恢复操作。进入可信恢复模式后，CRTM 阶段安装的 Recovery

Protocol 会被执行,负责从 USB 设备中读入除 CRTM 之外的可信固件的其他部分的映像,并写回到 flash 芯片中。可信恢复的过程同样需要对被加载的固件映像进行可信度量,因此要求操作者提供完整合法的恢复映像。

每一次可信度量,度量请求者都需要记载日志,并且通过 TPM_Extend 操作将日志的 Hash 值累计存储到 TPM 中指定的 PCRs 中。

8.2.3 可信度量加密计算的实现

可信度量的 Hash 操作由 CRTM 提供统一的实现代码,采用 SHA-1 算法生成 160bit 的 digest。从安全要求来考虑,Hash 操作由 CRTM 提供实现或由 TPM 提供实现并没有任何安全差别,从实现性能考虑,对于 1Mbits 的信息,UTBIOS 提供的 Hash 实现平均用时小于 13ms,而 SINOSUN SSX35B TPM 芯片提供的 Hash 实现平均用时小于 258ms[112]。

可信度量签名的解签操作由 TPM 完成,采用密钥长度为 2048 位的 RSA 非对称加密算法。因为解签操作所需要的公钥存储在 TPM 中,而对于激活的密钥,TCG 规范规定必须要求在 TPM 中密封计算,以防密钥泄漏。解签操作得到的 Hash 值返回给 CRTM,由 CRTM 将其与新计算的 Hash 值进行比较。

8.2.4 特殊处理

现有的 Legacy Option ROM 和 OS Loader Code 在系统最初执行的时候未被可信封装,需要在系统首次运行时对其作可信初始化处理,处理方法为:系统初次可信初始化时,由 UTBIOS 自动生成一对用于外部实体验证的密钥;UTBIOS 计算外部实体的 Hash 值并使用私钥对实体的 Hash 值进行签名,签名后抛弃私钥;利用存储固件的 flash 芯片中 NVRAM 数据区保存外部实体的数字签名,并将对应公钥导入到 TPM 中保护存储。第一次可信初始化完成后,以后每次启动就可实现对外部实体的可信测量。

存在多种类型的 OS Loader Code 产品，目前 UTBIOS 只针对 MBR 进行可信度量。

8.3　CTC 与 PDI 划分

CTC 与 PDI 的合理划分对可信固件实现的安全性至关重要，划分不合理则会破坏可信固件的安全性，或者破坏可信固件的实用性。

UTBIOS 在实现中，PDI 的划分只针对固件的动态配置数据，而将外部的其他静态数据项，如封装在固件 flash 芯片中的静态数据块，则等同于代码处理，通过可信度量测试这种静态数据块的完整性。

8.3.1　CTC 的封装形式

UTBIOS 中绝大多数 CTC 存储在固件 flash 芯片中。固件 flash 芯片中对应的内容称为固件映像（Firmware Image）。Intel Framework 定义了一种固件文件系统描述固件映像的存储格式[113,114,115]。

一个固件映像通常由一个或多个固件卷（Firmware Volume，FV）排列构成，固件卷由一个或多个固件文件系统（Firmware File System，FFS）封装而成，固件文件系统由一个或多个节（Section）封装而成。节可以是包装节（Encapsulation Section），也可以是叶子节（Leaf Section），包装节中还可包含节。典型的固件卷的结构如图 8.2 所示。

叶子节中包含的数据存在多种类型，可能是固件的数据块，如 EFI_SECTION_GUID_DEFINED、EFI_SECTION_VERSION、EFI_SECTION_DXE_DEPEX 等类型，也可能是可执行代码，如 EFI_SECTION_PE32、EFI_SECTION_TE 类型的结构性代码和 EFI_SECTION_COMPATIBILITY16 类型的非结构性代码。节中的数据或代码允许压缩或非压缩存储。UTBIOS 通过对固件映像的遍历操作访问和加载固

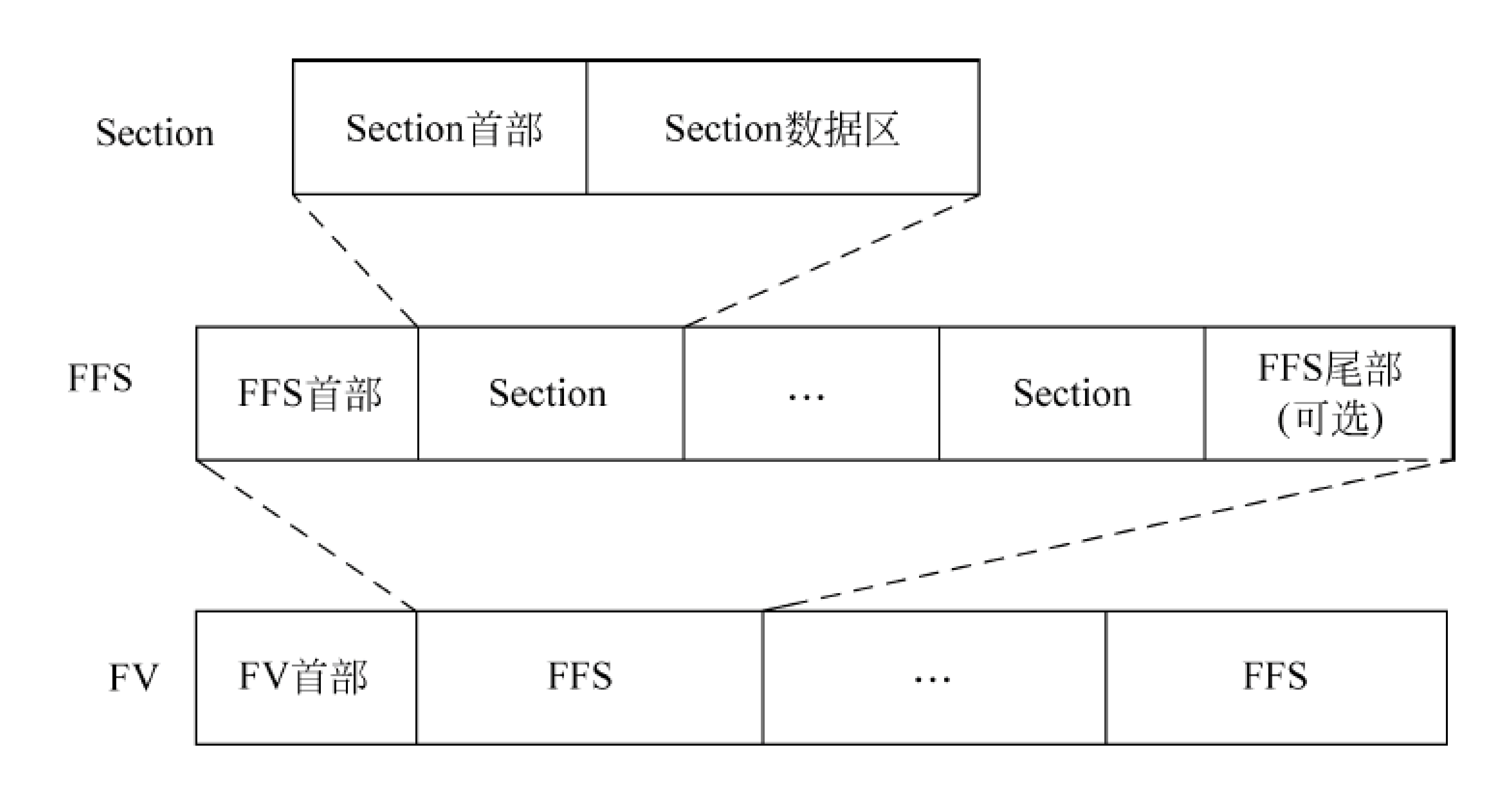


图 8.2　固件卷结构

件卷中的数据和代码节。

固件映像的模块化结构为 UTBIOS 的可信封装提供了很大的灵活性。对于固件映像中的 CTC(无论是静态数据块还是代码块),允许以多种单位形式进行封装。最小的封装单位确定为叶子节。尽管可以更小,如以叶子节中的结构性代码为封装单位,但是考虑到叶子节中甚至有非结构性的数据和代码,更深层次的细分并不适合程序的实现。最大的封装单位则可以是一个固件卷,即对整个固件卷进行完整信验证。

固件映像之外加载的 EFI 应用程序或驱动通常是以.efi 为扩展名的 PE/COFF 格式的可执行文件。这时候可以直接以 PE/COFF 可执行文件作为一个可信封装的单位。

扩展卡或控制器中的 OPROM 也需要作为一个独立的可信封装单元,为适应可信度量的需要,应该为这种传统 OPROM(Legacy OPROM)定义新的 OPROM 的格式。UTBIOS 在处理传统 OPROM 时,目前按照 8.2.4 节的方法做特殊处理。

8.3.2　CTC 的划分

可信固件保护环模型通过将可执行代码单元划分为 CTC 和 OTC 来限制代码对受保护数据的写操作,实现了对代码行为的局部限制和监控。CRTM 包含的代码不

受此种划分限制，或者说，CRTM 包含的代码全部都属于 CTC 类型。在 UTBIOS 中，CTC 和 OTC 的划分主要针对 DXE 及其之后阶段的代码。flash 映像中包含的代码大部分都是 EFI 固件实现所必需的，只有少数属于应用级的代码，而更多的应用级的代码往往在计算机集成商和用户使用阶段被嵌入或加载。这就决定 flash 芯片映像中包含的代码大部分为 CTC 代码。

表 8.1 分类列出 UTBIOS 中的 CTC，而 UTBIOS 中目前包含的 OTC 包括：文件系统支持(FAT16/FAT32)、EFI Shell、TCP/IP 协议、Socket Lib、EFI Shell Commands 等。

表 8.1　UTBIOS 包含的 CTC

代码作用域	包含的模块/协议
框架代码	DxeIpl，Bds，DxePlatform
flash 芯片操作支持	FwBlockService
系统管理模式支持	SmmAccess，SmmControl，SmmBase，SmmRelocate
计数器与定时器	MonotonicCounter，SmartTimer，WatchDogTimer
芯片组和控制器	DxeIchInit，Ich7SmmDispatcher，GmchMbi，IdeController，SataController，DiskIo
内存管理	LightGenericMemoryTest，MemorySubClass，MemoryInit
总线支持	LightPciBus，IdeBus，IsaBus，PciHostBridge，PciPlatform
USB 标准支持	Ehci，Uhci，UsbBus，Usbkb，UsbMassStorge
控制台	GraphicsConsole，Terminal，VgaClass，BiosVideo，BiosKeyboard
传统 BIOS 兼容支持	Legacy8259，LakeportLegacyRegion，IntelIchLegacyInterrupt，LegacyBios，Compatibility16，LegacyFloppy
SMBIOS 标准支持	Smbios，SmbiosMisc
ACPI 标准支持	AcpiS3Save，AcpiSupport，AcpiPlatform
BIOS 设置管理程序	SetupBrower
处理器微代码补丁	MicroCode

8.3.3　PDI 的划分

根据可信固件保护环模型对受保护数据项 PDI 和不受保护数据项 NDI 的划分原则，UTBIOS 对系统运行时的动态数据进行分类划分，对 PDI 进行访问存储限制和

保护。

动态数据总体上可分为两类：一类是固件根据对系统硬件检测的结果报告的信息，这一类通常划分为 NDI；另一类是允许程序或用户根据需要对系统软硬件进行配置的数据项，这类数据项根据其重要性可分别划分为 PDI 或 NDI。

表 8.2 和表 8.3 分别列出了 UTBIOS 中主要的 NDI 和 PDI，UTBIOS 实现中的 NDI 和 PDI 包含但不限于表中所列的内容，表中所列内容为后续实现提供参考。

表 8.2 UTBIOS 中的 NDI

数据项中文名称	数据项英文名称	备注
BIOS 版本	BIOS Version	CRTM 自动检测
BIOS 序列号	BIOS ID	CRTM 自动检测
主机厂商信息	System Message	CRTM 自动检测
处理器类型	Processor Type	CRTM 自动检测
处理器主频	Processor Clock Speed	CRTM 自动检测
处理器总线速度	Processor Bus Speed	CRTM 自动检测
处理器二级缓存总量	Processor L2 Cache	CRTM 自动检测
处理器序列号	Processor ID	CRTM 自动检测
处理器是否支持多核	Multiple Core Capable	CRTM 自动检测
处理器是否支持超线程	Hyper-threading Capable	CRTM 自动检测
处理器是否支持 64 位计算	64bit Technology	CRTM 自动检测，PC 机通常为 EM64T
内存总量	Installed Memory	CRTM 自动检测
内存速度	Memory Speed	CRTM 自动检测
内存通道模式	Memory Channel Mode	CRTM 自动检测
x 号 PCI 插槽使用状态	PCI Slot x	显示该插槽是否占用
x 号并口配置	LPT Port x	传统接口，很少用，不会影响系统启动
x 号串口配置	Serial Port x	传统接口，很少用，不会影响系统启动
省电模式	Low Power Mode	Off/On
挂起模式	Suspend Mode	S3/S1
快速启动	Fast Boot	只影响系统启动速度
数字键锁定	Numlock Key	
日期/时间	Date/Time	
语言选择	Language Select	UTBIOS 支持中文/English

表 8.3 UTBIOS 中的 PDI

数据项中文名称	数据项英文名称	备　注
引导顺序	Boot Sequence	Usb/cdrom/SATA/PATA/NIC 设置不当系统不能引导
软盘驱动器设置	Diskette Drive	floppy/usb/embedded 设置不当磁盘不能使用
SATA-x 驱动器设置	SATA-x	Off/on 设置不当磁盘不能使用
PATA-x 驱动器设置	PATA-x	Off/on 设置不当磁盘不能使用
IDE 模式设置	IDE Mode	Compatible/enhanced 设置不当磁盘不能读写
SATA 模式设置	SATA Mode	IDE/RAID 设置不当磁盘不能读写
板载网卡设置	Onboard NIC	Off/on/PXE 设置不当系统不能引导
板载声卡设置	Onboard Audio	Off/on 设置不当声卡不能使用
USB 控制器设置	USB Controller	Off/on/no boot USB 已经成为系统启动过程中的重要设备
USB 端口设置	Front USB Ports	Off/on
主视频设备设置	Primary Video	Auto/Onboard 设置不当系统不能引导，无显示
超线程设置	Hyper-Threading	Off/on
超频设置	Speed-Step	Off/on
自动开机设置	Auto Power on	随意更改导致安全隐患
自动开机时间设置	Auto Power Time	随意更改导致安全隐患
远程开机设置	Remote Wake up	随意更改导致安全隐患
装载缺省参数	Load Defaults	可能导致正常参数丢失
管理员密码	Admin Password	安全属性
用户密码	User Password	安全属性
TPM 使能	TPM Disable	安全属性
TPM 激活	TPM Deactivated	安全属性
TPM 属主	TPM Owner	安全属性

8.4 UTBIOS 安全设置与日志

8.4.1 BIOS Setup 程序安全与 TPM 设置

UTBIOS 在用户初次使用时，并不立即启动可信度量过程。用户必须通过 BIOS Setup 程序界面对 TPM 进行设置，并执行适当的初始化工作，以后的启动过程才会

启用可信度量操作。

根据保护环模型的强制性实施规则 MER4 的安全要求,用户进入 BIOS Setup 需要进行身份验证。UTBIOS 目前采用密码验证,由于 UTBIOS 对 C 语言程序运行的完全支持,以及对 USB 协议和高级图形功能等的完全支持,使用 Ukey 或指纹验证的技术手段较之传统 BIOS 实现难度也大大降低,基于这种更安全验证技术的开发工作也即将展开。

在 BIOS Setup 程序中,允许安全管理员 SA 对 TPM 选项进行设置。BIOS Setup 程序中对 TPM 的设置项包括:

使能状态:Enable/Disable

激活状态:Activated/Deactivated

属主状态:Owned/Not Owned/Clear

使能状态和激活状态主要的区别在于:在 Disable 状态下,TPM 不可以做 Owner 的设置操作,而在 Deactivated 状态下则没有此项限制。必须在 SA 完成 TPM 的属主操作后,UTBIOS 才会启用可信度量机制。

TPM 通过 LPC 总线连接到南桥芯片上,通过 LPC 总线访问 TPM 标识寄存器映射空间不需要做任何初始化工作,因此 UTBIOS 在 CRTM 的较早阶段就可以通过访问 TPM 厂商标识寄存器读取厂商标识[116],判断系统启动过程中是否存在可用的 TPM 物理硬件,以决定是否激活 BIOS Setup 程序中的 TPM 安全选项,以及是否在启动过程中启动可信度量。

8.4.2 可信度量日志

UTBIOS 在每一次可信度量后,会产生可信度量日志。可信度量日志记录的结构包括下面的数据项[117,118,119]。

- PCRIndex:事件 Digest 扩展存储到 TPM 此编号的 PCR 中。
- EventType:事件类型,TCG 对不同阶段的事件类型在文档[118]中定义。
- Digest:Event 的 Hash 值,在完整性报告中验证事件数据的完整性。

- EventDataSize：Event 数据的长度。
- Event：事件描述数据。

UTBIOS 不直接在外部存储设备上存储可信度量日志，而是申请一片 ACPI 保留内存，将日志记录保存在保留内存中，通过 ACPI Table 的指针传递给操作系统使用，这同 TCG 的相关规范也是相吻合的[118]。日志记录必须按照可信度量事件发生时间顺序排列。无论是日志记录的内容被破坏，还是记录的顺序被破坏，都将造成 PCRs 的完整性报告结果的失败。

8.5　可信度量的性能分析

UTBIOS 同 Legacy BIOS 相比较，在性能上主要受可信度量的影响。可信度量主要作 hash 生成和签名校验操作。以 SHA-1 hash 算法和密钥为 2048b 的 RSA Verification 操作来估量可信度量对固件启动时间的影响。RSA 加密操作（即 Signature）需要占用大量时间[120]，但加密过程只在生产过程中使用，UTBIOS 正常运行过程中可信度量则只有解密操作。

表 8.4 列出了在 1GHz Pentium PC 上和在 SINOSUN SSX3B TPM 中 SHA-1 操作和 RSA 解签操作所花费的时间[112,120]。

表 8.4　PC 与 TPM 加密算法时间比较

算　　法	密钥长度 b	操　　作	PC 耗费时间(s)	TPM 耗费时间(s)
SHA-1		Hash(1Mbits)	<0.013	<0.258
RSA	1024	Verification	<0.094	<0.015
RSA	2048	Verification	<0.098	<0.040

UTBIOS 的可信度量时间可以按下式来计算：

$$T = T(L1) + T(L2)$$

$$T(L1) = \sum_{i=1}^{n} t(m_i) = \sum_{i=1}^{n} (t(H(i)) + t(V(i)))$$

$$T(L2) = t(H(L2)) + t(V(L2))$$

T(L1)是 UTBIOS 对 BIOS 内部实体和外部加载的 EFI Driver、EFI Application 以及 Option ROM 进行可信度量消耗的总时间。T(L2)是 UTBIOS 对 OS Loader Code 进行可信度量消耗的时间。$t(m_i)$是对第 i 个实体进行可信度量的时间。t(H())代表生成 SHA-1 digest 的时间，t(V())代表签名校验的时间。

UTBIOS 可信度量的 SHA-1 操作由 PC 完成，RSA Verification(2048b)操作由 TPM 完成。则对一个 1Mb 的实体 i 可近似取得：

$$t(H(i)) = 0.013s$$

$$t(V(i)) = 0.040s$$

即对一个 1Mb 的实体的可信度量约耗费 0.053s。UTBIOS 中被测量的实体个数通常小于 100 个，实体 size 通常小于 0.5Mb，因此可得 T(L1)约为 2.7s。OS Loader Code size 通常小于 10Mb，则可近似取得：

$$t(H(L2)) = 0.13s$$

$$t(V(L2)) = 0.40s$$

则 T(L2)约为 0.57s，T 约为 3.27s。实际上，由于 UTBIOS 中需要测量的实体个数和 size 通常远小于这里用于计算的最大值，因此实际的计算时间也要远小于这个时间。

本书在采用 Pentium 3.0GHz 的实验中实际测得不做可信度量的 BIOS 启动时间约为 11s，而允许可信测量后固件启动时间约为 16s。实际测得的可信度量消耗时间大于计算估值，这其中包含了设备访问时间和数据交换时间。

8.6 本章小结

在本书的工作之前，国际国内尚未见有从理论方法到技术实现的整体安全设计的安全固件研究和产品报道。本章以前两章对安全 BIOS 的安全策略模型和可信度量方法研究结果为指导，以 TCG 相关规范为依据，研究一种可信固件——UTBIOS

产品原型开发的关键技术和实现，重点对最小 CRTM 构造的约束条件和组成、核心可信代码的划分、受保护数据项的划分及实现进行了阐述。

UTBIOS 产品原型，是在中国电子科技集团公司信息化工程总体研究中心的“太行安全 BIOS”项目产品基础上实施开发的。该项目受到信息产业部电子产业发展基金“新一代安全 BIOS 的研制和产业化”、“高安全与可管理 BIOS”项目的支持，其平台硬件相关的 BIOS 资料和代码得到 Intel 公司的授权和技术支持。正是基于这样的新一代 EFI/UEFI BIOS 项目和产品支持，才使本书可信固件研究的产品原型得以顺利实现。

Computer Firmware Security Technology

第 9 章

结　论

一直以来，操作系统、应用级安全以及安全芯片一直是计算机系统和网络信息安全领域研究的重点，而计算机在操作系统运行之前的启动和引导阶段，即固件 BIOS 运行阶段的安全性则长期被忽视。本书从多起与固件 BIOS 相关的安全事件的分析入手，指出固件 BIOS 在计算机系统安全中的重要性，以及对操作系统安全的基础保障作用。

本书介绍并分析了固件 BIOS 产品研发现状和研发技术，开展了对固件 BIOS 安全问题的系统性研究工作。本书的研究涵盖固件 BIOS 系统安全问题的两个方面：

（1）分析现存的大规模应用的传统 BIOS 产品存在的安全漏洞，以及由此引起的针对 BIOS 系统自身，以及操作系统和应用的安全威胁和实现技术，研究对计算机 BIOS 系统进行安全检测的方法，通过安全检测和维护增强现有 BIOS 系统的产品安全性。

（2）结合可信计算需求，定位固件 BIOS 在可信计算终端中的作用和地位，研究安全固件系统开发的策略模型、实现方法和关键技术，开发实现可信固件的产品原型。

9.1 本书研究的主要成果

本书围绕固件BIOS系统安全问题的两个方面开展深入研究，主要取得了以下5个方面的研究成果：

（1）以市场现有传统BIOS固件产品为对象，首次比较系统地分析并验证了计算机BIOS固件系统存在的安全漏洞，对漏洞特征和漏洞表示进行研究，建立了第一个计算机BIOS漏洞库。

（2）分析了多起典型BIOS固件安全威胁事件，分析概括了操作系统与BIOS固件系统的依赖性和安全互动性，提出并实现一种新型BIOS固件木马技术。该技术防可用于安全代理，攻可用于嵌入式木马、病毒或rootkit。

（3）研究了对计算机BIOS固件进行安全检测的原理和技术方法，提出基于漏洞扫描和恶意代码检测的计算机BIOS固件安全检测模型，开发实现了一种计算机BIOS固件安全检测系统。该系统能够对目前市场上主流BIOS固件产品进行安全检测，目前已经成功地进行了产品转化。

（4）通过考较不同历史阶段对可信与安全两个概念的使用情况，阐明可信与安全两者之间的区别和联系。本书认为，可信的本质是行为的安全，TCG完整性度量并不总是能够保障系统的安全性，只有行为的可信度量才等价于安全。分析了固件BIOS的安全需求，给出了可信固件的定义，提出适合计算机固件系统的可信固件保护环安全策略和模型。

（5）通过对TCG完整性度量和报告、信任链构建等规范要求的分析，提出基于可信注册、可信封装、可信验证三个步骤的可信固件可信度量方法；以新一代EFI/UEFI规范和Intel Framework代码为基础，开发实现了一个可信固件的产品原型——UTBIOS。在UTBIOS的开发实现中，对可信固件保护环模型实现的关键问题进行研究，给出了最小CRTM构造的约束条件和组成，给出了核心可信代码CTC和受保护数据项PDI的一个划分实现，设计了UTBIOS的可信度量

结构。

固件系统安全的重要性在过去很长一段时间内被忽视。本书首次建立了计算机BIOS固件漏洞库,开发设计并产品化一种计算机BIOS固件安全检测系统,提出可信固件保护环安全策略模型并实现了一种可信固件产品原型。本书这些创新性研究成果对于基于计算机底层固件BIOS系统的终端安全防护和可信终端构建的研究,具有积极的推动作用和借鉴意义。

9.2 进一步的研究方向

本书首次开展对计算机BIOS固件安全问题系统性的研究工作,研究内容涵盖了传统BIOS固件产品的安全保障和新一代安全固件产品开发两个方面,取得了预期的成果。从技术开发和理论研究来讲,下一步需要继续完善和深入探索的内容和方向包括:

(1) 计算机BIOS固件安全检测产品的进一步完善:包括BIOS固件漏洞库的持续跟踪研究和补充;BIOS固件模块样本的进一步充实;扩大可检测的BIOS固件产品类型和计算机产品类型的覆盖面。

(2) UTBIOS的开发,试图解决可信计算机固件BIOS运行阶段安全和信任链建立问题。完整的信任链建立,需要计算机其他方面的可信实现的同时跟进,包括可信硬件部件、可信OS Loader、可信操作系统以及可信应用软件的全面实施和紧密结合,这给可信计算的实施提出了更多更高的要求,也给后续的可信计算的研究指出了更广阔的研究方向。

(3) 正如本书所指出的,可信的本质是行为的可信而不是关系的可信,可信的度量也应该是行为的度量而不仅仅是完整性和真实性度量。为了能够在实体执行前或执行中度量、监控和限制实体的行为,本书认为基于语言的行为逻辑验证和检测的理论方法有着很好的前景,其他的行为度量的理论方法也是值得探索的领域。

（4）现有的计算机系统的安全体系，将安全保障者和被保障者混淆于同一执行环境中，安全保障者自身的安全得不到有效保障，而被保障者对系统性能的要求也往往被侵犯，从而导致安全处于一种死循环的困境。本书认为可以利用嵌入式技术为计算机系统在操作系统之外构造一种带外系统，利用带外系统有效解决这一问题，下一步可展开对这一问题的研究。

参考文献

[1] 陈文钦.BIOS Inside：BIOS 研发技术剖析.台湾地区：旗标出版股份有限公司，2001

[2] Compaq，Phoenix，Intel. BIOS Boot Specification v1.01. 1996.11

[3] Phoenix Technologies Ltd. Developer's Reference PhoenixBIOS 4.0 Release 6.1. 2000

[4] 杨柳.计算机安全：封堵 BIOS 漏洞.瞭望新闻周刊，2004，(19)：52-53

[5] Intel. Extensible Firmware Interface Specification Version 1.10. 2002.12

[6] UEFI Forum. Unified Extensible Firmware Interface Specification Version 2.0. 2006.1

[7] UEFI Forum. Unified Extensible Firmware Interface Specification Version 2.1. 2007.1

[8] Adam Agnew，Adam Sulmicki，Ronald Minnich，William Arbaugh. Flexibility in ROM：A Stackable Open Source BIOS. the Proceedings of the USENIX Annual Technical Conference (FREENIX Track). 2003.6

[9] Adam Sulmicki. LinuxBIOS：A Study of BIOS Services as used by Microsoft Windows XP. The Maryland Information Systems Security Lab

[10] 石文昌. 安全操作系统开发方法的研究与实施. 博士论文，中国科学院软件研究所，2001.11

[11] 单智勇. 多安全政策支持框架研究及其在安全操作系统中的实践. 博士论文，中国科学院软件研究所，2002.11

[12] 赵庆松. 安全操作系统的恶意代码防御技术的研究与实施. 博士论文，中国科学院软件研究所，2002.11

[13] 刘海峰. 安全操作系统若干关键技术的研究. 博士论文，中国科学院软件研究所，2002.9

[14] 季庆光. 高安全级操作系统形式设计的研究. 博士论文，中国科学院软件研究所，2004.3

[15] 朱鲁华. 安全操作系统模型和实现结构研究. 博士论文，解放军信息工程大学，2007.7

[16] 谭良. 可信操作系统若干关键问题的研究. 博士论文，电子科技大学，2007.3

[17] Arbaugh W.A，Farber D.J，Smith J.M. A Secure and Reliable Bootstrap Architecture. Security and Privacy，1997. Procedings，1997 IEEE Symposium on 4-7 May 1997：65-71

[18] Drew Dean，Ed Felten，Dan Wallach. JAVA security：From HotJava to Netscape and beyond. In 1996 IEEE Symp. Security and Privacy. IEEE，1996.5

[19] G. Morrisett，D. Tarditi，C. Stone，R. Harper，P. Lee. The TIL/ML compiler：Performance and safety through types. In 1996 Workshop on Compiler Support for Systems Software，1996

[20] Greg Morrisett，David Walker，Karl Crary，Neal Glew. From System F to typed assembly language. In 1998 Symposium on Principles of Programming Languages. IEEE，1998.1

[21] George C. Necula. Proof-carrying code. In Proc. 24th Symp. Principles of Programming Languages. ACM，1997.1

[22] George C. Necula，Peter Lee. Safe kernel extensions without run-time checking. In Proc. 2nd Symp. Operating System Design and Implementation. ACM，1996.10

[23] Dexter Kozen. Efficient Code Certification. Technical Report98-1661，Computer Science Department，Cornell University，1998.1

[24] Adelstein F，illerman M，Kozen D. Malicious Code Detection for Open Firmware. Computer Security Applications Conference，2002，Proceedings 18th Annual 9-13 Dec.2002：403-412

[25] Matt Stillerman，Dexter Kozen. Demonstration：Efficient Code Certification for Open Firmware. Proceedings of the DARPA Information Survivability Conference and Exposition，IEEE，2003

[26] John Heasman. Implementing and Detecting an ACPI BIOS Rootkit. Blackhat DC，2006

[27] John Heasman. Implementing and Detecting an PCI Rootkit. Blackhat DC，2007

[28] Nation Science and Technology Council. Federal Plan for Cyber Security and Information Assurance Research and Development. 2006.4

[29] Intel. 英特尔英保通技术及软件白皮书 v1.1. 2005

[30] Intel. Intel Active Management Technology Deployment and Reference Guide Version 1.0. 2006.10

[31] Intel. Intel® Centrino® Pro and Intel® vPro™ Processor Technology. 2007.7

[32] TCG. TCG Specification Architecture Overview Specification Revision 1.4. 2007.8

[33] 国家密码管理局. 可信计算密码支撑平台功能与接口规范. 2007.12

[34] 张焕国，毋国庆，覃中平等. 一种新型安全计算机. 武汉大学学报(理学版)，2004，50(S1)：1-6

[35] 黄强，沈昌祥. 可信计算技术对操作系统的安全服务支持. 武汉大学学报(理学版)，2004，50(S1)：15-18

[36] 方艳湘，黄涛. Linux 可信启动的设计与实现. 计算机工程，2006，32(9)：51-53

[37] 黄涛，沈昌祥. 一种基于可信服务器的可信引导方案. 武汉大学学报(理学版)，2004，50(S1)：12-14

[38] 李晓勇，韩臻，沈昌祥. Windows 环境下信任链传递及其性能分析. 计算机研究与发展，2007，44(11)：1889-1895

[39] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1：Basic Architecture. 2006.9

[40] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2：Instruction Set Reference. 2006.9

[41] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3：System Programming Guide. 2006.9

[42] Intel. Intel Platform Innovation Framework for EFI Architecture Specification Version 0.9. 2003.9

[43] IEEE. Boot (Initialization Configuration) Firmware：Core Requirements and Practices Revision / Edition：94. 1994

[44] Intel. Intel 945G/945GZ/945P/945PL Express Chipset Family Datasheet. 2005.12

[45] Intel. Intel I/O Controller Hub 7(ICH7) Family Datasheet. 2006.1

[46] Intel. Intel Desktop Board D945GNT Technical Product Specification. 2005.5

[47] PCI Special Interest Group. PCI Local Bus Specification Revision 2.2. 1998.12

[48] PCI-SIG. PCI Express Base Specification Revision 1.0. 2002.7

[49] Ravi Budruk，Don Anderson，Tom Shanley. PCI Express 系统体系结构标准教材. 田玉敏，王崧，张波译. 北京：电子工业出版社，2005

[50] HP，Intel，Microsoft，Phoenix，Toshiba. Advanced Configuration and Power Interface Specification Revision 3.0b. 2006.10

[51] DMTF. System Management BIOS (SMBIOS) Reference Specification Version 2.5. 2006.9

[52] Intel. Compatibility Support Module Specification Revision 0.96. 2006.4

[53] Compaq，Phoenix，Intel. Plug and Play BIOS Specification Version 1.0A. 1994.5

[54] Stefan Einarsson，Marvin Rausand. An Approach to Vulnerability Analysis of Complex Industrial Systems. Risk Analysis. 1998，18(5)：535-546

[55] Phoenix Technologies Ltd.. PhoenixNet 1.4 PRD Revision 0.6. 2000.6

[56] SANS Institute. NSA Glossary of Terms Used in Security and Intrusion Detection. 1999

[57] Intel. Intel Platform Innovation Framework for EFI ACPI Specification Draft for Review Version 0.9. 2004.4

[58] Intel. Intel Platform Innovation Framework for EFI ACPI Table Storage Specification Draft for Review Version 0.9. 2004.4

[59] Phoenix Technologies Ltd.. Standard BIOS 32-bit Service Directory Proposal Revision 0.4. 1993.6

[60] Phoenix Technologies Ltd.. PhoenixBIOS 4.0 Revision 6 User's Manual. 2000.1

[61] Intel. Intel 82802AB/82802AC Firmware Hub(FWH) Datasheet. 2000.11

[62] SHANKAR U，TALWAR K，FOSTER J S，et a1. Detecting format string vulnerabilities with type qualifiers. USENIX Security Symposium，USA. 2001

[63] 袁新哲，胡昌振，戴斌. Windows NT/2000 典型漏洞特征分析及知识表达方法. 探测与控制学

报，2003，25(增刊)：67-70
[64] 乔佩利，赵文军，赵妍．CVE 数据库及其在入侵检测中的应用．哈尔滨理工大学学报，2004，9(4)：69-72
[65] MANN D E，CHRISTEY S M．Towards a Common Enumeration of Vulnerabilities．Presented at 2nd Workshop on Research with Security Vulnerability Databases，Purdue University．West Lafayette，IN，1999
[66] BAKER D W，CHRISTEY S M，Hill W H，et a1．The Development of a Common Enumeration of Vulnerabilities and Exposures．the Second International Workshop on Recent Advances in Intrusion Detection，1999
[67] Tim Lindholm，Frank Yellin．The Java virtual machine specification．Addison Wesley，1996
[68] M．Abadi，R．Stata．A type system for Java bytecode subroutines．In Proc．25th Symp．Principles of Programming Languages．ACM SIGPLAN/SIGACT，1998.1：149-160
[69] Robert O'Callahan．A simple，comprehensive type system for Java bytecode subroutines．In Proc．26th Symp．Principles of Programming Languages．ACM SIGPLAN/ SIGACT，1999.1：70-78
[70] George C．Necula．Compiling with proofs．PhD thesis，Carnegie Mellon University，1998.9
[71] George C．Necula，Peter Lee．The design and implementation of a certifying compiler．In Proc．Conf．Programming Language Design and Implementation．ACM SIGPLAN，1998：333-344
[72] George C．Necula，Peter Lee．Safe，untrusted agents using using proof-carrying code．In Giovanni Vigna，editor，Special Issue on Mobile Agent Security，volume 1419 of Lect．Notes in Computer Science．Springer-Verlag，1998.6：61-91
[73] N．Glew，G．Morrisett．Type-safe linking and modular assembly language．In Proc．26th Symp．Principles of Programming Languages．ACM SIGPLAN/ SIGACT，1999.1：250-261
[74] G．Morrisett，K．Crary，N．Glew，D．Grossman，R．Samuels，F．Smith，D．Walker，S．Weirich，and S．Zdancewic．TALx86：A realistic typed assembly language．In Proc．Workshop on Compiler Support for System Software．ACM SIGPLAN，1999.5：25-35
[75] Sabin T．Comparing binaries with graph isomorphisms．2004．http://razor.bindview.corn/publish/papers/comparing-binaries.html
[76] Halvar Flake．Structural comparison of executable objects．In DIMVA，2004：161-173
[77] Dullien T，Rolles R．Graph-based comparison of executable objects．2005．http://www.sabre-security.com/files/BinDiffSSTIC05.pdf
[78] Ware，W．H．，ed．Security Controls for Computer Systems：Report of Defense Science Board Task Force on Computer Security，AD # A076617/0，Rand Corporation，Santa Monica，Calif．，

1970，reissued 1979

[79] CSC-STD-001-83，Department of Defense Standard. Department of Defense Trusted Computer System Evaluation Criteria. DoD Computer Security Center，1983. 8

[80] DoD Directive 5200. 28，Security Requirements for Automatic Data Processing (ADP) Systems，1978. 4

[81] DoD 5200. 28-M， ADP Security Manual-Techniques and Procedures for Implementing，Deactivating，Testing，and Evaluating Secure Resource-Sharing ADP Systems，1979. 6

[82] Anderson，J. P. Computer Security Technology Planning Study，ESD-TR-73-51，vol. I，ESD/AFSC，Hanscom AFB，Bedford，Mass.，1972. 10 (NTIS AD-758 206)

[83] Bell，D. E. and LaPadula，L. J. Secure Computer Systems：Mathematical Foudations. Hanscom AFB，Bedford，MA，Rep. FSD-TR-73-278，vol. 1，ESD/AFSC，1973

[84] Bell，D. E. and LaPadula，L. J. Secure Computer Systems：A Mathematical Model. Hanscom AFB，Bedford，MA，Rep. FSD-TR-73-278，vol. 2，ESD/AFSC，1973

[85] Bell，D. E. and LaPadula，L. J. Secure Computer Systems：Unified Exposition and Multics Interpretation，MTR-2997 Rev. 1，MITRE Corp.，Bedford，Mass.，1976. 3

[86] Trusted Computer Systems Evaluation Criteria，US DoD 5200. 28-STD，1985. 12

[87] Canadian Trusted Computer Product Evaluation Criteria，Version 3. 0，Canadian System Security Centre，Communications Security Establishment，Government of Canada，1993. 1

[88] Information Technology Security Evaluation Criteria，Version 1. 2，Office for Official Publications of the European Communities，1991. 6

[89] Federal Criteria for Information Technology Security，Draft Version 1. 0(Volumes Ⅰ and Ⅱ)，jointly published by the National Institute of Standards and Technology and the National Security Agency，US Government，1993. 1

[90] ISO/IEC 15408：1999. Information Technology-Security Techniques-Evaluation Criteria for IT Security. 1999

[91] 屈延文. 软件行为学. 北京：电子工业出版社，2004

[92] National Computer Security Center. Glossary of computer security terms. NCSC-TG-04，1988. 10

[93] Denning D. E. A Lattice Model of Secure Information Flow. Communications of the ACM，1976，19(5)：236-243

[94] Goguen J. A，Meseguer J. Security policies and security models. In：Proc. of the 1982 IEEE Symposium on Security and Privacy. IEEE Computer Society Press，1982，4：11-20

[95] Goguen J. A，Meseguer J. Unwinding and Inference Control. In：Proc. of the 1984 IEEE

Symposium on Security and Privacy. IEEE Computer Society Press, 1984,5: 75-86

[96] K. J. Biba. Integrity considerations for secure computer systems. Technical Report, No. ESD-TR-76-372, Electronic Systems Division, Air Force Systems Command, 1977

[97] Clark, D. D. , Wilson, D. R. , Comparison, A. : A Comparison of Commercial and Military Computer Security Policies. In: Proceedings of the 1987 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos, 1987

[98] Dr. David F. C. Brewer and Dr. Michael J. Nash. The Chinese Wall Security Policy. 1989 Symposium on Security and Privacy, IEEE, 1989. 5

[99] W. E. Boebert, R. Y. Kain. A Practical Alternative to Hierarchical Integrity Policies. In proceedings of 8th National Computing Security Conference, Gaithersburg, 1985. 10

[100] R. O'Brien, C. Rogers. Developing Applications on LOCK. In Pro. 14th National Computer Security Conference, Pages 147-156, Washington, DC, 1991. 10

[101] R. Sandhu, P. Samarati. Access Control: Principles and Practice. IEEE Computer, 1994,9: 40-48

[102] R. S. Sandhu, et al. Role-Based Access Control Models. IEEE Computer. IEEE Press, 1996, 29(2): 38-47

[103] Harrison M, Ruzzo W, UIlman J. Protection in operating systems. Communications of the ACM, 1976,19(8): 461-471

[104] T. Mayfield, J. Roskos, S. Welke and J. Boone. Integrity in Automated Information Systems. National Computer Security Center(NCSC), Technical Report, 1991: 79-91

[105] TCG. TPM Main Specification Version 1. 2 Part 1 Design Principles. 2007. 7

[106] TCG. TPM Main Specification Version 1. 2 Part 2 TPM Structures. 2007. 7

[107] TCG. TPM Main Specification Version 1. 2 Part 3 Commands. 2007. 7

[108] TCG. TCG Credential Profiles Specification Version 1. 1 Revision 1. 014. 2007. 5

[109] TCG. TCG PC Specific Implementation Specification Version 1. 1. 2003. 8

[110] Intel. Intel® Platform Innovation Framework for EFI Pre-EFI Initialization Core Interface Specification (PEI CIS) Version 0. 91. 2004. 11

[111] Intel. Intel® Platform Innovation Framework for EFI Driver Execution Environment Core Interface Specification (DXE CIS) Version 0. 91. 2004. 12

[112] Sinosun. SSX35B Trusted Platform Module (TPM) Version 1. 2. 2006. 9

[113] Intel. Intel Platform Innovation Framework for EFI Firmware File System Specification Version 0. 9. 2003. 9

[114] Intel. Intel Platform Innovation Framework for EFI Firmware Volume Specification Version 0. 9. 2003. 9

[115] Intel. Intel Platform Innovation Framework for EFI Firmware Volume Block Specification Version 0.9. 2003.9

[116] Sinosun. SSX35 TPM BIOS Porting Guide Version 1.01 revision 0. 2006.3

[117] TCG. TCG PC Client Specific Implementation Specification for Conventional BIOS Version 1.20 Revision 1.00. 2005.7

[118] TCG. TCG EFI Platform Version 1.20 Revision 1.00. 2006.6

[119] TCG. TCG EFI Protocol Version 1.20 Revision 1.00. 2006.6

[120] Menasce D A. Security Performance[J]. IEEE Internet Computing,2003,7(3): 84-87

致谢

由于偶然的机会，作者有幸加入到国内 BIOS 固件安全研究的第一批队伍中，通过十多年的工作研究，有幸取得了一点成果。有感于国内固件安全研究方面多年以来一直的缺失，缺乏关注度，难得找到一本系统性的相关著作和参考资料，因此，心中时常有要把自己的研究工作写出来，能与有共同兴趣爱好者、从业者们共享。犹豫彷徨，又受老师、朋友和同业者们的时常鼓励，终于还是决定要义无反顾地完成这篇著作，即使常常汗颜于自身的水平和取得的非常有限的研究成果，但也算是对我国的计算机固件安全研究的一种抛砖引玉了。这本著作，沁透着作者十多年研究的汗水，也饱含着身边许多人对我的期望和热忱帮助。在本书即将封笔完成之际，我发自内心地对多年来关心、支持、帮助我的老师们、同学们、同事们、朋友们、亲人们说一声：谢谢你们！

首先，我要衷心地感谢我尊敬的导师许榕生研究员。是许老师为我创造了最好的学习机会和条件，多年来在生活和学习中给予我无微不至的关怀和悉心的指导，鼓励我、指引我在科学研究的道路上不断前进。许老师学识渊博、治学严谨、思维敏捷、诲人不倦，在治学、工作和为人处世等各方面都令我深受教益，终生不忘。

感谢我的亲如兄长的朋友王军和刘军，是他们最早引导我走上固件安全的研究之路，并为此克服种种困难，提供了研究所需的软硬件环境和条件。

感谢中科院高能所计算中心刘宝旭研究员，感谢他在项目研究过程中的无私帮助和指导，感谢他对我工作和生活上的关心和支持。本书所取得的成果和贡献，同他的无私帮助是分不开的。

感谢北京电子科技学院的方勇院长、池亚平教授对本书研究工作及 BIOS 安全检测和认证项目的大力支持和帮助。

感谢中国电子科技集团公司信息化总体研究中心的李微微主任、李铭博士对本书研究工作的大力支持和帮助，他们的支持为本书可信固件产品原型的实现提供了新一代 EFI/UEFI 平台和代码基础。

感谢参与和承担本书项目研究相关任务的师兄弟和同事们，正是因为他们的协助与支持，才使本书项目研究的实验和系统开发得以顺利完成。他们包括：中科院高能所计算中心和网络安全实验室的杨泽明、卢志刚、吴焕、于磊等；北京电子科技学院的周伟东、吴丽军、孙春燕等；福州大学的陈楣、李云峰等；燕山大学的赵文华、梁志国、王猛等；中国电子科技集团公司信息化总体研究中心的陈小春、张超、朱立森等。

感谢沈阳工程学院李卓玲教授对本书出版的友情支持。

特别地，我要感谢我的爱妻马瑶，多年来她默默地承担了家庭重担，支持我走过这段研究历程。谢谢我淘气可爱的女儿周艾丽，她给我带来了无穷的欢乐和精神动力。

在本书写作过程中，还有很多给予我关心、支持和鼓励的善良的人们，在此我无法一一列出他们的名字，但我内心充满对他们的感激和尊敬。在这里，请他们接受我真诚的谢意和祝福。

周振柳

2012 年 7 月　沈阳